

“Implementation and Secure Extension of Mo Soft Core in FPGA.”



Author

Quratulain Abdul Ghafoor

Fall –MS2016 (CE) 00000171601

Supervisor

Dr. Farhan Hussain

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
ISLAMABAD

April 2020

“Implementation and Secure Extension of Mo Soft Core in FPGA.”

Author

Quratulain Abdul Ghafoor

Fall 2016-MS (CE) 00000171601

In partial fulfillment of degree requirement

MS Computer Engineering

Supervisor of Thesis

Dr. Farhan Hussain

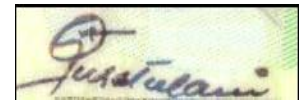
Signature of thesis supervisor: _____

COMPUTER ENGINEERING DEPARTMENT
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD

Declaration

Its certified that research work whose title is “Implementation and secure extension of Mo soft core in FPGA” is my own work. I haven’t presented that work anywhere for assessment; neither I have taken material from other resources.

The contents that I have referenced from other resources are properly acknowledged.

A rectangular box containing a handwritten signature in blue ink. The signature appears to be 'Quratulain' with a stylized initial 'Q'.

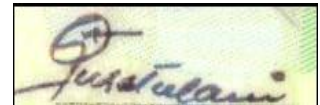
Student Signature

Quratulain Abdul Ghafoor

Fall 2016-MS (CE) 00000171601

Language Correctness Certificate

This thesis is thoroughly read by English expert and is free of grammatical, typing, syntax, spelling mistakes and semantic. The thesis is in accord of the format provided by university.

A rectangular box containing a handwritten signature in black ink. The signature appears to be 'Quratulain' written in a cursive style.

Student Signature

Quratulain Adul Ghafoor

Fall 2016-MS (CE) 00000171601

Supervisor Signature

Dr. Farhan Hussain

Copyright Statement

- The copyright on the thesis belongs to the author of thesis. Copies (by any means), in whole or in part, may only be according to the instructions given to the students and filed with the NUST, College of Electrical & Mechanical Engineering Library. Librarian can provide the details. This page is must to be part of all copies which are made. Other copies (by any means) cannot be made without the written permission of the author.
- The ownership of all intellectual property rights described in thesis belongs to NUST College of Electrical & Mechanical Engineering, subject to any prior agreement, and will not be available to third party without the written consent of the owner. College of E & ME, prescribe the terms and conditions of said agreement.
- Additional information at which exploitation and disclosures may take place on the conditions available at the NUST College Library of Electrical & Mechanical Engineering, Rawalpindi.

Acknowledgement

All praise and glory to the Almighty God (most glorified, the highest) who blessed me the courage, patience, knowledge and ability to carry out that work and persevere and carry it out satisfactorily.

Without a doubt, he has given me the way and without HIS blessings, I cannot achieve anything.

I am obliged to my advisor, Dr. Farhan, for raising the spirit of my spirit and for his continued support, motivation, dedication and valuable advice in my quest for knowledge. I am highly thankful to Allah having a cooperative advisor and a kind mentor for my research.

With my advisor, I thank all my dissertation committee: Dr. Muhammad Usman Akram, Dr. Arslan Shaukat of CEME and Fahad Alghazali, Muazzam shehzad, Dr. Asad of NESCOM for their cooperation and cautious suggestions.

Finally, I express my special gratitude to my family, colleagues and to the people who have encouraged and supported me during this period.

“Dedicated to those who care for mankind for sake of Allah.”

Abstract

In Electronics World processors are brains and can be implemented as soft core at FPGAs or ASICs for specific applications. M0 core developed by ARM because of its specifications and speed is a widely used general purpose processor in industrial and military applications. M0 core because of its upward compatibility, RISC architecture and utilizing the 12000 logic gates in its small configuration makes it very feasible to be implemented for various tasks. Because of its low power rating low its use in low power devices is very much appropriate. . We can utilize M0 core as a co-processor providing communication services, crypto graphical services, malware detection services, wireless sensor nodes etc. ARM Cortex M0 core is a propriety processor and has a few limitations of its own. As a propriety core the details of its architectures are not published. Also interfacing of the memory (RAM, ROM) and UART needs to be done to use it for a specific application. In our work we comprehensively study the architecture of M0 core. We thoroughly investigated the Bus (AHB Lite) of M0 core in order to interface it with the RAM and ROM. We were able to successfully interface the Bus with the memories. Four Blocks of 8x2048 Bits RAMs are designed. We also interfaced the Peripheral bus (APB) for UART. UART communication is implemented. Our experiment demonstrated successful transmission of data. For our experiment we used the Xilinx Spartan 3-E kit to port Cortex M0 processor on FPGA. Xilinx FPGA XC3S1600E with 33,000 logic gates are enough to port Cortex M0 soft core. In order to visualize results Digital Clock Manager (DCM) is used to scale down 50 MHz clock frequency to 5MHz. and practically readable data is coming at HyperTerminal with variable baud rate according to requirement showing practical implementation of coprocessor communicating successfully.

List of Abbreviations

UART	Universal Asynchronous Receiver Transmitter.
FPGA	Field Programmable Gate Arrays.
BRAM	Buffered Random Access Memory.
ROM	Read Only Memory.
ABP	Advanced Peripheral Bus.
TTL	Transistor Transistor Logic.
DCM	Digital Clock Manager.
SRAM	Static Random Access Memory.
WSN	Wireless Sensor Node.
NIC	Network Interrupt Controller.
ASIC	Advanced System Integrated Chip.
DSP	Digital Signal Processor.

Table of Content:

I.	Declaration	3
II.	Language Correctness Certificate	4
III.	Abstract	8
IV.	Table of Content:	10
V.	List of Tables	14
Chapter 1: INTRODUCTION		15
1.1	Advantages of FPGAs	15
1.1.4	ARM series of processor	16
1.2	Problem Statement	17
1.3	Objectives and Aims.....	18
1.4	Contributions.....	18
1.5	Structure of Thesis	18
Chapter 2: LITERATURE REVIEW		19
2.1	Embedded Systems and FPGAs.....	19
2.2	Architecture of Microcontroller:.....	19
2.3	FPGA and soft cores of Processors	21
2.4	ARM Understanding Different Types of Processor	22
2.5	ARM Cortex M0.....	26
2.6	M0 advantages:	29
2.7	Applications:	30
Chapter 3: Hardware and Software Details		31
3.1	FPGA Spartan-3E as proto Board	31
3.2	Hardware overview.....	32
3.2.1	Slide switches	32
3.3	Clock.....	35
3.4	FPGA Configuration Options	36
3.5	Serial ports	37
3.6	FPGA programming.....	38
Chapter 4: Proposed Methodology		42
4.1	M0 processor	43
4.2	Slave	43
4.3	Slave select.....	43

4.4 Address decoding.....	44
4.5 APB Bus	44
4.6 ROM	44
4.6 SRAM.....	45
4.7 ABP Slave select	45
4.8 ABP Timer.....	46
4.9 ABP UART	46
Chapter 5: UART communication	48
5.1 M0 processor	48
5.2 Serial vs. Parallel	48
1. Asynchronous Serial.....	49
2. Synchronous Serial.....	49
5.5 Wiring and Hardware.....	51
5.6 UARTs (Universal Asynchronous Receiver Transmitter)	52
5.8 Common Pitfalls.....	54
Chapter 6: EXPERIMENTS AND RESULTS	56
6.1 Simulation	56
6.2 Implementation	59
Chapter 7: EXPERIMENTAL OUTPUT RESULTS ON HARDWARE	63
7.1 Results.....	63
Chapter 8: CONCLUSION and FUTURE WORKS	64
8.1 Conclusion	64
8.2 Future Work	64
References	65

List of figures

Figure 1-1 ARM Microcontroller Cores.....	16
Figure 2. 1 Basic microcontrollers Architecture.....	19
Figure 2.2 Trade-off in processor designs.....	22
Figure 2. 3 ARM processor families Overview.....	23
Figure2.4 Instruction table of M family.....	25
Figure 2.5 Cortex M series processor compatibility.....	26
Figure2.6 Cortex M0 simplified block diagram.....	27
Figure2.7 Cortex M0 simple systems.....	28
Figure3.1: Xilinx Spartan FPGA Board	31
Figure 3.2: Four Slide Switches.....	32
Figure 3.3: UCF for Slide Switches.....	32
Figure 3-4: Rotary push buttons and four push buttons.....	32
Figure 3.5: Push buttons must have internal pull down resistor in FPGA.....	33
Figure 3.6: UCF for Push-Button Switches.....	33
Figure 3.7: Pull up resistor for push button switch in FPGA.....	33
Figure 3.8: UCF file for Rotary Push-Button Switch.....	34
Figure 3.9: Eight surface mount LEDs.....	34
Figure 3.10: UCFs for Eight Discrete LEDs.....	34
Figure 3.11: Available Clock.....	35
Figure 3.12: UCF for Clock Sources.....	35
Figure 3.13: Configuration settings of Spartan-3E Starter Kit for FPGA.....	36
Figure 3.14: Detailed Configuration Options.....	37
Figure 3.15: RS-232 Serial Ports.....	37
Figure 3.16: UCF for DCE RS-232 Serial Port	38
Figure 3.17: iMPACT open up option.....	38
Figure 3.18: New configuration file for FPGA.....	38
Figure 3.19: iMPACT Programming Succeeded	39
Figure 3.20: Set Properties for Bit stream Generator.....	39
Figure 3.21: PROM, ACE, or JTAG File generation.....	40
Figure 3.22: Click PROM File Formatter.....	40
Figure 3.23: Selection of XCF04S Platform Flash PROM.....	40
Figure 3.24: PROM File Formatter Succeeded.....	41

Figure 4.1 Block diagram of M0.....	42
Figure 4.2: 'cmsdk_ahb_default_slave.v' model.....	43
Figure 4.3: 'cmsdk_ahb_slave_mux.v' model	43
Figure 4.4 cmsdk_ahb_rom model.....	44
Figure 4.5 cmsdk_ahb_to_sram.v model.....	45
Figure 4.6 cmsdk_apb_slave_mux.v module.....	45
Figure 4.7 Block diagram of timer.....	46
Figure 4.8 Block diagram of ABP UART.....	46
Figure 5.1 parallel communication.....	47
Figure 5.2 serial communication.....	47
Figure 5.3 data framing.....	49
Figure 5.4 data packet for serial protocol.....	50
Figure 5.5 Hardware connection.....	51
Figure 5.6 5volt TTL signal.....	51
Figure 5.7 13 volt TTL signal.....	52
Figure 5.8 UART interface.....	52
Figure 5.9 UART internal block diagram.....	53
Figure 5.10 Pro mini design.....	54
Figure 5.11 Mismatched data.....	54
Figure 6.1 Keil Program.....	56
Figure 6.2 FPGA settings.....	57
Figure 6.3: Path for image.hex file in cmsdk_ahb_rom.v	57
Figure 6.4: Path for image.hex file in cmsdk_fpga_rom.v.	58
Figure 6.5: Path for BRAM files in cmsdk_fpga_rom.v.....	58
Figure 6.6: Simulation of M0 core processor.....	59
Figure 6.7: Path for image.hex file in cmsdk_ahb_rom.v.....	59
Figure 6.8: BRAM settings.....	60
Figure 6.9: 8x2048 size BRAM.....	60
Figure 6.10: BRAM file path.....	60
Figure 6.11: DCM clock frequency settings.....	61
Figure 6.12: UCF of M0 core	61
Figure 6.13: top level M0 core implementation.....	62
Figure 6.14: M0 core implementation on FPGA.....	62
Figure 7.1: Result data.....	63

List of Tables

Table 2.1: Typical components in microcontrollers.....	21
Table 3.1 Clocks, Global buffers and DCM.....	36

Chapter 1: INTRODUCTION

Motivation

Susceptible attacks and hacking of processor architectures, theft of cryptographic keys led this research work to start. For that an appropriate, low power, high speed, efficient and cost effective solution is investigated for architectural design. Fast speed processor core in FPGA researched and investigated to be the brilliant idea because of its flexibility, design modification and customization properties.

In the age of technology, to be competitive, the product must have dynamic and unexpected changes. In this work, field programmable gate matrices (FPGA), in the age of technology, are enough to show the dynamic and unexpected changes of the product adapted to meet these requirements. They are economical, adaptable, powerful and economical and can be configured in a hardware using HDL language.

Traditionally, the designer can perform has following three options to perform hardware control platforms:

- DSP (Digital signal processors).
- ASIC (Application specific integrated circuits).
- FPGA (Field programmable Gate arrays).

ASICs provide optimal performance because they are designed to meet the requirements of the application. However, the budgets for ASIC-based solutions are maximized, since production machines is usually relative small due to the small number units compared to the products it produces. On the other hand, DSP-based solutions are economical, but their processing speed is not sufficient due to software execution.

1.1 Advantages of FPGAs

The development of FPGA technology has grown in recent years in terms of usable resources, processing speed, number of access points, retail prices and energy consumption. Advanced FPGAs are therefore a reliable alternative to common microcontrollers and ASICs for applications. Recent FPGA-based systems also include many advantages of DSP and ASIC. That is, high flexibility, reuse capacity, rapid development cycles, moderate costs and easy updating (due to the use of the abstract hardware description language (HDL)), and feature expansion

(provided FPGA resources are not depleted) in addition, current FPGAs can be integrated as software processors. Therefore, an FPGA can have typical processor capabilities.

1.1.1 FPGAs as ARM microprocessors:

Primary use of an FPGA is for computationally intensive, parallel processing tasks and high-speed, however the ARM microprocessor is used mainly due to its versatility many manufacturer used it as the core of their applications i.e ZYNQ family of devices. ARM because of their wide use as a processor variations of operating systems are available.

1.1.2 Embedded processors on FPGA soft-core vs hardcore

In integrated systems, field programmable gate array (FPGA) is a very effective solution due to their ability to reconfigure, scale and be low cost. Due to the configurable logical capability of the FPGA, designers have been forced to integrate software processors into the FPGA for use in various peripherals. To this end, many software and hardware cores have been developed for different software and hardware.

1.1.3 Linux soft cores

The cores of the arm processor are called cortex. The Cortex-M FPGA solutions provided by Altera (soft cores of M1 is provided) also Actel provide the same package, while the Cortex-M3 is in available as IC package. Xilinx has provided no practical solution to date. ARM has released a low-cost and low-cost solution of the M0 processor that can be programmed in an FPGA or used as ASIC applications. XILINX does not provide a soft MB core.

1.1.4 ARM series of processor

The Cortex series of processors as shown in Figure 1, includes cores of economical microcontroller solutions and state of the art processors which are capable of handling huge tasks and advanced operating systems. Other processors of the same family are ARM7 series, ARM9 series and ARM11 series. The specialized SecurCore™ series is also primarily intended for security as well as cryptography applications.

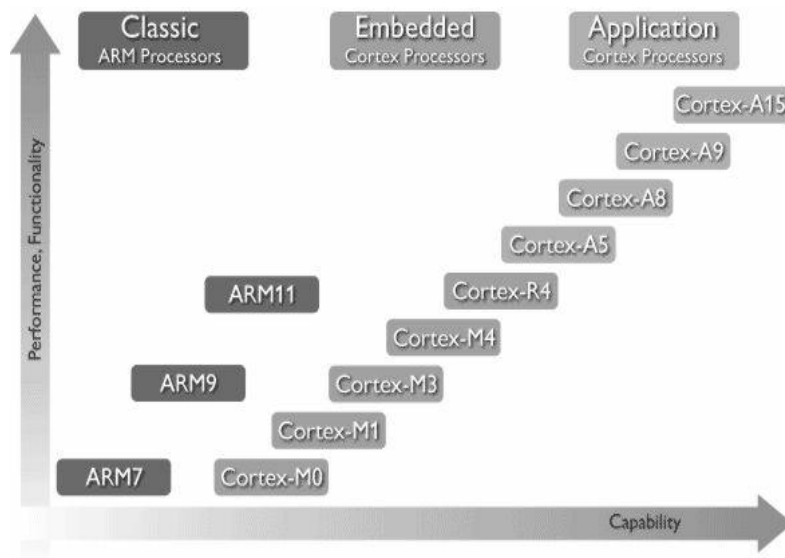


Figure 1-1 ARM Microcontroller Cores

1.1.5 Advantages of ARM processor family:

ARM cortex processors mainly used for high performance applications with real time and embedded operating systems. These processors are low cost, low power, fixed latency interrupt handling processors. Cortex Mo is alternative for 8 bit microcontrollers as they have high processing capacity.

1.1.6 Motivation

In the field of embedded systems, a hardware which can be configured and changed even after installation and deployment is much needed to cope with the ever increasing and changing demand of market. The motivation behind this thesis is to develop indigenous hardware platform for various industrial and military configurable systems and devices with low power, high interrupt latency, good efficiency and low cost.

This design or platform can be used as IOT, Encryption controller, intrusion detection controller and as communication control (for implementing various communication protocols).

The most attractive feature of this platform is its speed, low cost and run time configuration feature.

1.2 Problem Statement

To develop an indigenous Mo core based on FPGA, hardware configurable solution for various communications, industrial and military applications with all features of ARM processor in form of soft-core of ARM family.

1.3 Objectives and Aims

The main goals of the investigation are the following:

- It can be implemented as indigenous IOTs (Internet of Things)¹.
- It can also be used as parallel intrusion detection processor like PHANTOM².
- Can be used as platform for computer vision applications³.
- Can be used to make FPGA based multiprocessor systems⁴.
- Secure FPGA soft core processors for symmetric key cryptography.⁵
- As coprocessor for specific need in various applications.

1.4 Contributions

- Comparison of working of soft-core and hardcore ARM processors implementations.
- Selection of FPGA instead of ASIC and DSP processors.
- FPGA Spartan 3 board all functionalities and features.

1.5 Structure of the Thesis

This work is structured as under:

Chapter 2: States literature review of related work.

Chapter 3: Give details about the Hardware used in this thesis.

Chapter 4: Consists of the proposed methodology in detail. It includes: coding, system configuration, simulations, results and analysis.

Chapter 5: This chapter consists of all the details, theory and implementation of UART communication.

Chapter 6: Closes the thesis and reveals the future prospect of this study.

Chapter 7: Experiments and results are discussed in detail with all desired figures and tables.

Chapter 8: Closes the thesis and reveals the future prospect of study.

Chapter 2: LITERATURE REVIEW

2.1 Embedded Systems and FPGAs

In embedded system industry FPGAs use is very crucial and vital. FPGA offers possibility of a wide range of electronic gates and IC to be incorporated.

Processors main use in electronic like cell phones, computers, and television, machines used for washing purposes and in various cards. And even in simple devices contain processor in them like remote control of your radio, which is actually fabricated in an IC called microcontroller .In modern microcontrollers, memory system and other peripherals are also incorporated.

2.2 Architecture of Microcontroller:

Typical elements of a microcontroller are as under mentioned

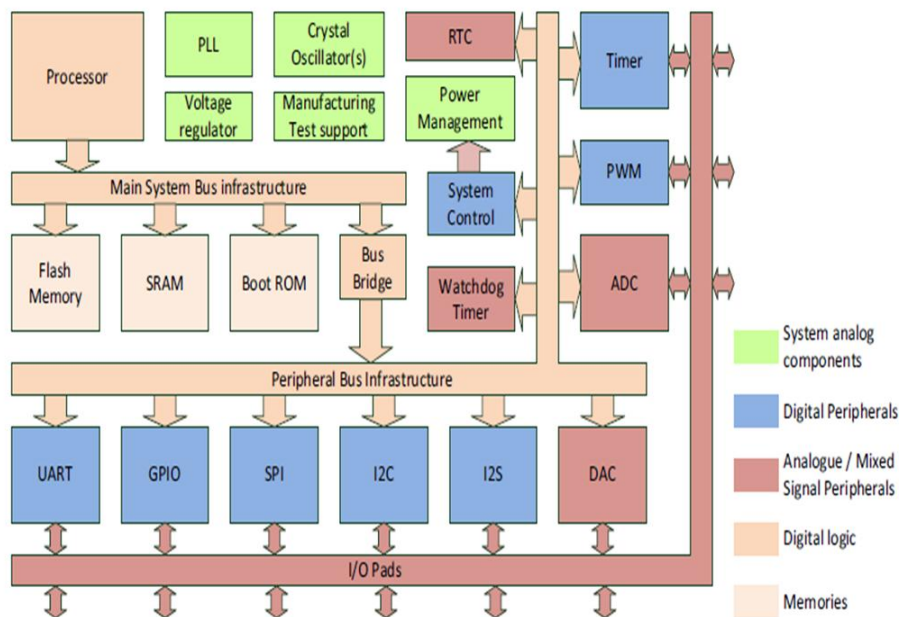


Figure 2. 1 Basic microcontrollers Architecture

Items	Descriptions
Read Only memory (ROM)	It is a nonvolatile memory used for storage of program.

Flash	Flash is a ROM that can be reprogrammed several times, typically used for storing program codes.
Static Random Access memory (SRAM)	Used for data storage.it is a form of volatile memory.
Phase Lock Loop (PLL)	Its primary used for generation of variable clock frequency from a clock (reference).
Real Time Clock (RTC)	It is simply a low power timer to calculate seconds and works with a low power oscillator.
General purpose Input/output (GPIO)	It is a sub module with parallel data interfacing and used for controlling external modules and to read back the external signal status.
Universal Asynchronous Receiver and Transmitter (UART)	It's a sub module used to cater serial transfer of data to external interface.
Inter Integrated Circuit (I ² C)	It is an interface used for serial data transfer; its data transfer rate is higher than serial.
Serial Peripheral Interface (SPI)	It's commonly used for serial communication with other off chip deices.
Inter IC sound (I ² S)	It's commonly used for serial communication for audio signal.
Pulse Width Modulator (PWM)	It is a module to output configurable square signal of flexible time duration.
Analog to Digital Convector (ADC)	Its primary purpose is to transform analog signal to digital format.
Digital to analog convector (DAC)	Its primary function is to convert digital signal form to analog signals.
NBatch Dog Timer	It's a programmable time device to ensure that processor is running as program. In enable state, the timer should be updated by the programmer after certain time interval. When the program is crashed that timer gets expired and on the basis of this critical system interrupt or system reset is triggered.

Table 2.1: Typical components in microcontrollers

2.3 FPGA and soft cores of Processors

FPGA facilitated the incorporation of soft processor cores. Processors as soft core can be considered equivalent to "computer on a chip" or microcontroller. Core conglomerates a Central processing Unit, memory and peripherals on a single chip. Custom interfaces can also be accessed beyond the real standard integrated FPGA of integrated chip.

Many types of microcontrollers are available with many types of processors, peripherals and memory sizes in them which can be marketed in various packages in market. Processors are used in various applications so that using software system can be controlled and various features can be developed.

Cortex M0+ processors and ARM cortex M0 and mainly used as SOCs, microcontrollers, ASIC and ASSPs. And sometimes used in many subsystems.

Available reduced instructions set computer (RISC) architectures in market are:

1. ALTERA launched CPUs like NIOS and NIOS II.
2. Gaisler Research launched CPU like LEON2 and LEON3.

Microcontroller systems are now becoming increasingly advanced and demanded and can achieve greater performance to fulfill the requirements of more suitable functions and operations. Sometimes, they are costly; therefore, microcontroller systems are determined as designs of a chip only detected in high-performance 32-bit processors, which dominate the current market. Variety of peripheral are evolved after requirement rose from software development and pressure of costly standardization.

Industry designed ARM M series of processors to meet above challenges, now over 200 companies are producing these chips from standard core to meet all applications like sensor nodes to radio communication for IOTs.

With the access of microcontrollers to more and more people, the software will be available for you to see. Real-time operating systems (RTOS) have quickly become a recommended industry exercise and their use gaining importance for software engineering. The combination of these components in the industry presented a problem for those who were developers, who were in charge of the industry to reduce system costs and marketing success. Therefore, the architectures of Cortex-M processors coupled with CMSIS standard is basis of hardware and software standard.

2.3.1 Applications of soft-core processors:

Soft-core processors have many applications:

1. As Embedded machine learning processor for intrusion detection [5].
2. As a secure processor called PHANTOM in which practical oblivious computation is performed [6].
3. Design and development of a security coprocessor based on a chip system (SOC) and a program protection mechanism for wireless sensor nodes (WSN).[7]

2.4 ARM Understanding Different Types of Processor

Industry has designed many processors for various applications. ARM also designed different processor series for different applications. For server, a processor with high data rate is required. For battery applications, performance can be compromised in order to achieve low power. For high performance, processor needs to have more transistors as this is rule of physics. As frequency increases so as the power dissipation by the processor.

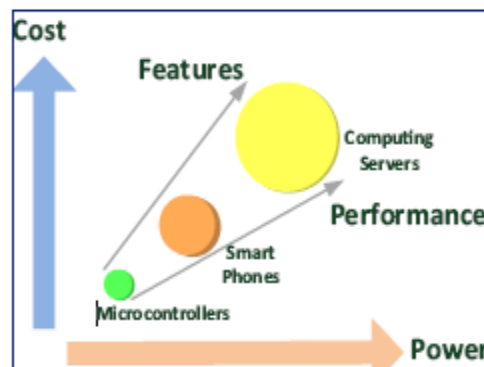


Figure 2. 2 Trade-off in processor designs

Increasing the size of silicon results in increases in production cost (Figure 2.2). Hence different processor have been designed for different applications. Also, chip designer's needs to select suitable processor for their application. Luckily, many vendors provide different processors for different applications. ARM also provides processors as per designer need.

2.4.1 Overview of the ARM Processor Families

ARM has designed alot processors for several applications as shown in the Figure 2.3. Let see what ARM has to offer for designers.

ARM designed the processor since long. Over 20 years of their experience ARM has provided 32 Bits processors but recently they have designed processor for mixed architecture of 32 Bits and 64 Bits.

ARM7 processor is the first series of processors marketed for the designers. It is high efficiency and high code density allows the designers to use state of the art operating system. They are used frequently in next generation mobile phones. ARM after great response from designers continue design new series like ARM9/9E and ARM11 family of processor.

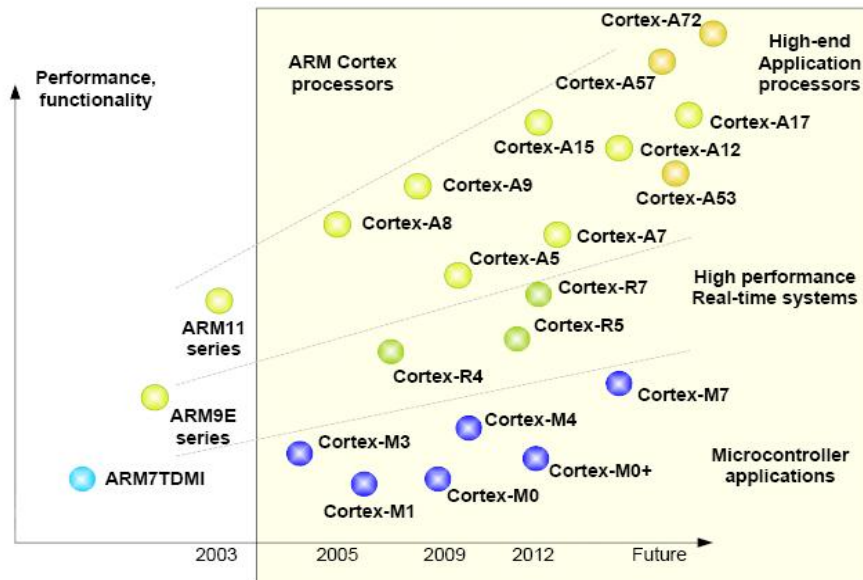


Figure 2. 3 ARM processor families Overview

2.4.1.1 Cortex-A series processors:

Some application requires high performance so that they support advance operating system. Such processors have longer processing channels and can operate at high clock frequency. (1 GHz or greater). For functionality MMU (Memory management unit) supports virtual memory addressing. These are used in cell phones, mobile computing and energy efficient processor. For fast response ARM has designed R series of processor.

2.4.1.2 Cortex-R series processors:

For fast response Cortex-R series is launched. Their clock frequency is less than A series (between 500 MHz to 1GHz). They have tightly coupled memories to improve response time. Some are provided with additional features like ECC (Error correction code) for reliability.

Their application includes disk drive controllers, wireless controllers/ modems, automotive controllers and in industrial controllers. They consume a lot of power and have complex architecture. For integrated products ARM launched another M series of processor.

2.4.1.3 Cortex-M series Processors:

M series is used for application where less processing is required at the cost of low power. Their pipeline is short (2 stage for M0 and 3 stage for M3, M4), however, M7 has pipeline of 6 steps because of greater performance requirements; however, it is much smaller than pipeline of high end application processors. Due to tube optimization and power consumption in the application, the maximum clock rates of them are slower than those of the R and A series processors. However, for low processing application this is not a issue.

Cortex-M series processors can handle very fast interrupts response. To achieve this goal they are equipped with special module called nested vector interrupt controller (NVIC). This module has very handy features of interrupt handling. They are easy in use and programmed in C language. Because of features like low power, high performance and user friendly Cortex-M series processors in sensors, chips used for mixed-signal ASIC / ASSP, and are even use as a controller in complex use processor / SoC product subsystems.

Instruction set is compared in figure 2.4, M0, M0+and M1 support small instruction set (56 instructions). Most are 16 Bits as they provide effiecient code density. M0 and M0+ instructions are simple in nature and can perform complex task easily. For state of the art operating system M3 processor is used as it has 32 bit of instruction support and can support following:

- More addressing modes of memory
- In the 32-bit instructions Larger immediate data
- Conditional branch ranges and longer branch
- Additional branch instructions
- Hardware divide instructions
- Multiply accumulate instructions (MAC)
- Processing instructions bit view
- Adjustment instructions and saturation.

M3 can handle complex data quickly. With same code size as M0 and M0+ because it uses same instructions for same task. 32 bit instructions can do same task efficiency then 16 bit instructions.

For DSP applications that involves filtering and signal transformation. M3 also equipped with SIMD (Single Instruction multiple data). Data path of processor is also reduce to accommodate that.

VSEL							VCVTA		VCVTN		VCVTF		VCVTM		VMAXNM		VMINNM		Cortex-M7 FPU (single and double precision floating point)										
VRINTR							VRINTA		VRINTN		VRINTP		VRINTM		VRINTX		VRINTY												
VABS			VADD			VCMP			VCMPE			VCVT			VCVTR			VDIV			VLDM			VLDR			Cortex-M4 FPU (single precision floating point)		
VMMLA			VMMLS			VMQV			VMRS			VMRSR			VMUL			VNEG			VMMLA			VMMLS			VMMAA		
VMNML			VPOP			VPUSH			VSQRT			VSTM			VSTR			VSUB			VFMA			VFMS			VFMAA		
QDADD			QADD			QADD16			QADD8			SADD16			SADD8			UADD16			UADD8			UHADD16			UHADD8		
QDSUB			QSUB			QSUB16			QSUB8			SSUB16			SSUB8			USUB16			USUB8			UHSUB16			UHSUB8		
ADC		ADD		ADR		AND		ASR		B		BIC		SHADD16		SHADD8		SHSUB16		PKH		SEL		SHSUB8					
BFC		BFI		CLZ		CDP		CLEX		CMN		CMP		SMULTT		SMULTB		SMULTB		SMULTT		SMULTB		UQADD16					
CBNZ		CBZ		DBG		EOR		LDR		LDRH		LDRB		LDRD		SMULBT		SMULBB		SMULBT		SMULBB		UQSUB16					
LDMIA		LDMDB		LDRT		LDRHT		LDRBT		LDRSH		LDRSB		SMLATT		SMLATB		SMLATT		SMLATB		SMLATT		SMLATB		UQADD8			
LDRSBT		LDRSHT		LDREX		LDREXB		LDREXH		LSL		LSR		SMLABT		SMLABB		SMLABT		SMLABB		SMLABT		SMLABB		UQSUB8			
LDC		MCR		MRC		MCRR		MRRC		PLD		PLI		SMLALTT		SMLALTB		SMLALTT		SMLALTB		SMLALTT		SMLALTB		SMMUL			
MOV		MOVW		MOVVT		MUL		MVN		MLS		MLA		SMLALBT		SMLALBB		SMLALBT		SMLALBB		SMLALBT		SMLALBB		SMMULWT			
NOP		PUSH		POP		ORR		ORN		PLDW		RBIT		USADAB		USADB		USADAB		USADB		USADAB		USADB		SMMULWB			
ADC		ADD		ADR		BKPT		BLX		BIC		REV		REV16		REVSH		ROR		QSAK		QASX		SMLAD					
AND		ASR		B		BX		CP5		CMN		RSB		RRX		SBC		SEV		UQSAX		UQASX		SMLSAD					
BL		MRS		MSR		SUB		STC		UBFX		SBFX		UASX		SASX		SMLALD		USAX		SSAX		SMLSAD					
DSB		DMB		ISB		STR		STRD		UDIV		SDIV		USAX		SSAX		SMLSLD		UHASX		SHASX		SMLAD					
CMP		EOR		LDR		LDRH		LDRB		LDM		STRB		STRH		UMULL		SMULL		UHASX		SHASX		SMLAD					
LDRSH		LDRSB		LSL		LSB		MOV		NOP		STMIA		STMDB		UMLAL		SMLAL		UHSAX		SHSAX		SMLUSD					
REV		REV16		REVSH		MUL		MVN		ORR		STREX		STREXB		UXTB		SXTB		UXTAB		SXTAB		SMLAWT					
PUSH		POP		ROR		RSB		SEV		SVC		STREXH		STRT		USAT		SSAT		UXTAH		SXTAH		SMLAWB					
SBC		STR		STRH		STRB		STM		SUB		STRHT		STRBT		UXTB		SXTB		UXTAH		SXTAH		SMLAWB					
SXTB		UXTB		SXTB		UXTB		TST		YIELD		TBB		TBH		WFI		WFE		UXTAB16		SXTAB16		SMLLA					
WFE		WFI		Cortex-M0/M0+/M1 (ARMv6-M)		TST		TEQ		YIELD		IT		UXTAB16		SXTAB16		SMLLA		UXTAB16		SXTAB16		SMLLA					
16-bit instructions							32-bit instructions							Cortex-M3 (ARMv7-M)							Cortex-M4 (ARMv7E-M)								

Figure 2.4 Instruction table of M family

2.4.1.4 Portability Software of Cortex®-M Processors:

M0, M0+ and M1 series processor are based on ARMv6. M3, M4 and M7 based on ARMv7 architecture as show in the figure 2.5 for instruction set support.

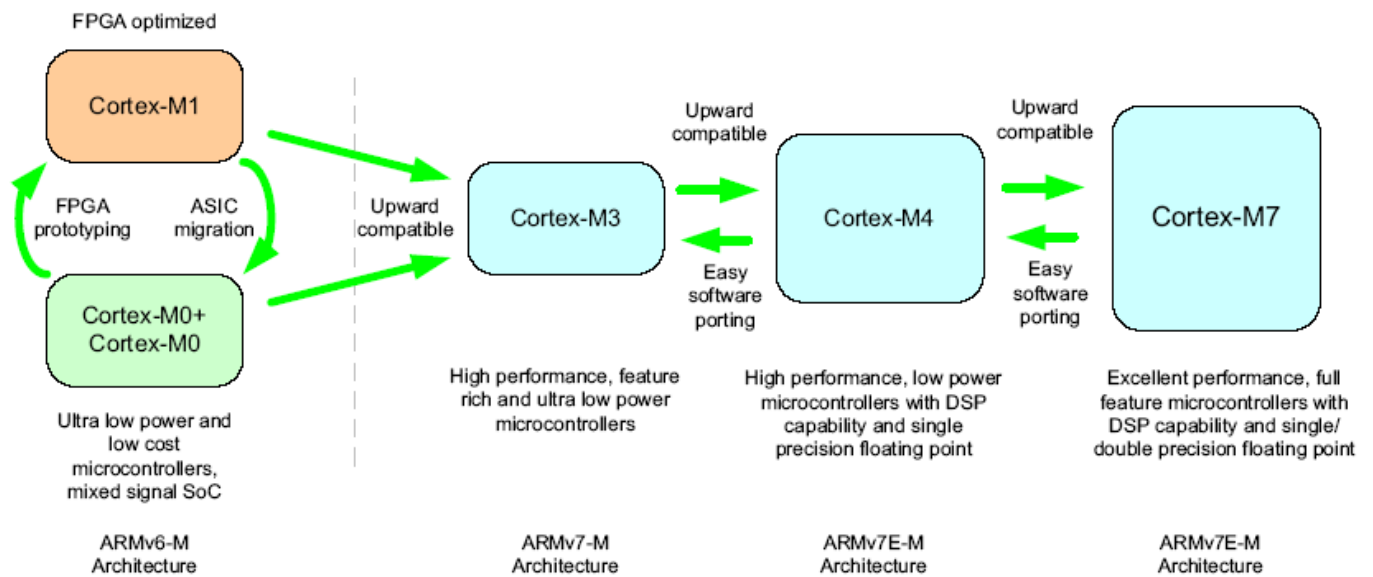


Figure 2.5 Cortex M series processor compatibility

2.5 ARM Cortex M0

It is 32 Bit processor; it means that internal registers banks, data path and bus interfaced are all 32 Bits. It has single bus interface means its architecture is Von Neumann type. It has three stage pipeline i.e. (Fetch, decode and execute). Most of the instructions are 16 bit, however only few are 32 bits. It can support optional 32x32 bit multiplier. Address supports 4GB of memory interfacing. M0 bus interface is based on AHB-Lite protocol that can support 8, 16 and 32 bit data transfer. Protocol is pipelined and can support high operating frequency and peripheral can be connecting AHB-Lite with APB Bus Bridge. M0 supports 32 interrupts through Nested Vector interrupt controller (NVIC). It supports Two sleep modes (normal sleep and deep sleep) are used for power saving.

2.5.1 Block Diagram:

Block diagram M0 is shown as figure 2.6

M0 contains register for data storage, ALU and control logic. Three stages of pipeline for fetch decode and execute stage. Banks are of 32 bits. Some are special usage like PC while others are used as general purpose.

NVIC can handle up to 32 interrupts with functionality to compare priority between interrupts request and current interrupt priority level. In case of a interrupt, NVIC communicate with the processor to execute interrupt correctly.

There are 32 bits AHB-Lite bus interface, processor core, internal bus system, and data path.

AHB-Lite bus

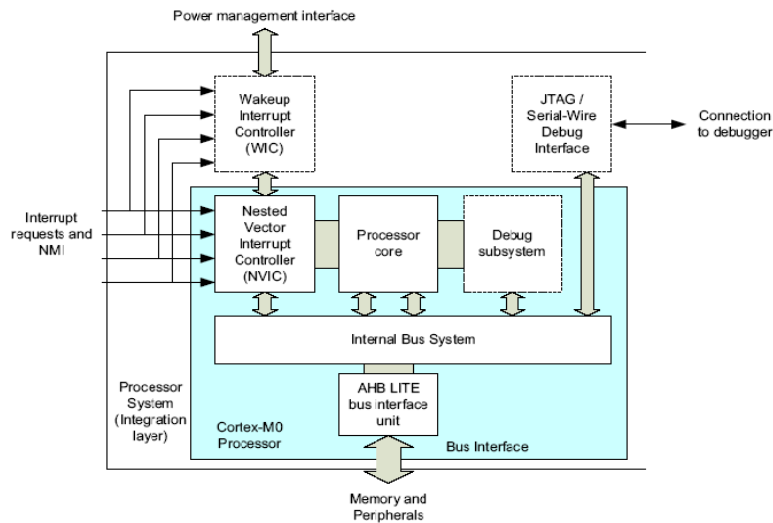


Figure 2.6 Cortex M0 simplified block diagram

2.5.2 System overview:

M0 processor inherently does not have any memory and peripherals. However, designers need them for their design, so M0 processors based IC have different addressing range, interrupts and memories. Normally M0 have following peripherals:

- Program code storage use ROM (Read only memory). e.g. flash memory
- SRAM (Static RAM) for data read write.
- Bus interfacing for various memories and processor joining.

M0 processor can be look like as shown in the Figure 2.7

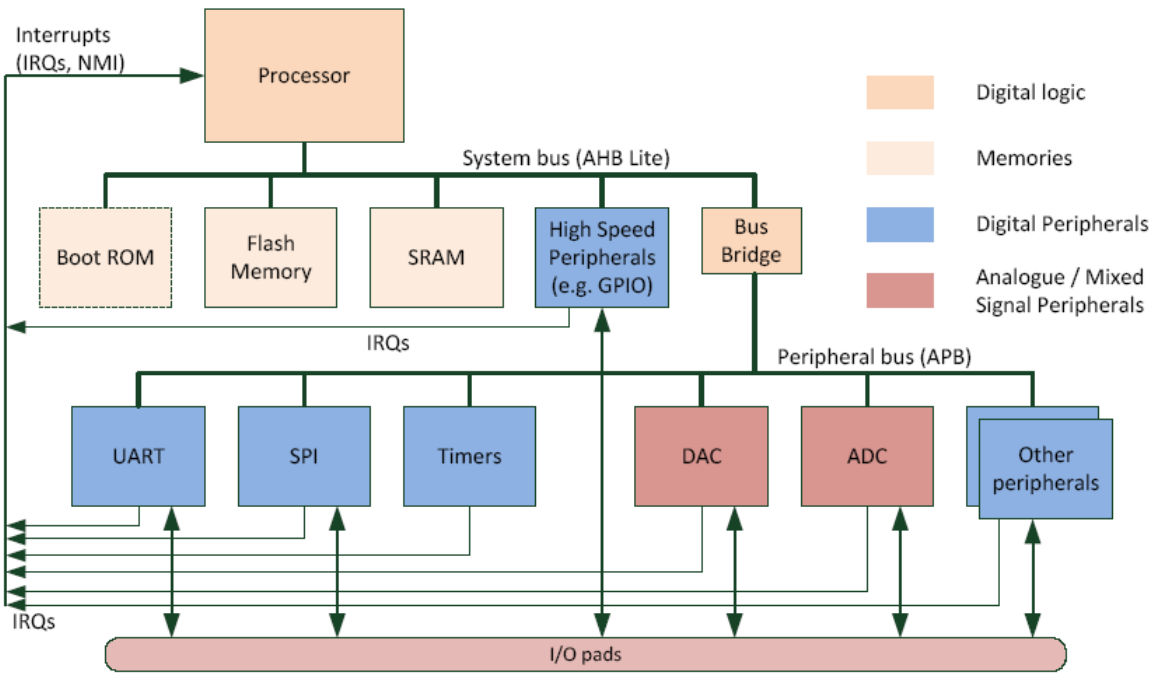


Figure 2.7 Cortex M0 simple systems

M0 based design might have bus partition into two parts:

- RAM, ROM, Flash memory, SRAM, few other peripherals connected by the bus and a bus bridge.
- Peripherals attached by the help of peripheral bus, might have different operation frequency.

APB is connected with AHB-Lite bus through Bus Bridge. APB bus use is as follows:

- Low cost solution as APB bus is non-pipelined and is simpler to implement than AHB-Lite bus.
- Makes possible to interface others sub modules at different frequency of operations than main bus.
- Where large combinational blocks are required for logic implementation, they are attached with APB bus to avoid bottle neck for high operating frequency.

Also, interrupts, GPIO (General Purpose Input/Output) modules can be handled easily through APB bus.

2.5.3 ARMv6-M Architecture

M0 cores are based on ARMv6-M Architecture. This refers to following two areas:

- ISA Model also called programmers model (software point of watching) and debug methodology (what debugger sees).
- Microarchitecture: An implementation detail like signal interfacing, execution sequence, pipelines etc. it is design specific. Like M0 has three stage of pipeline.

2.6 M0 advantages:

2.6.1 Low Power and Energy Efficiency

M0 is very energy efficient and consumes 12.5 uW/MHz with size of 90 nm. This is low power for 32 bit processor. ARM achieves it with less gate High efficiency count and Low power and through logical cell enhancement.

2.6.1.1 Low gate count

Low gate count is achieved through careful design techniques at each stage. Every part was developed very carefully and helps in minimizing gate count to 12000 only. This is lower than even 16 bit processor keeping the performance almost double.

2.6.1.2 High efficiency

M0 has many low power features to use in battery powered applications. Two modes sleep and deep sleep are available. Sleep mode is invoked using instructions like WFE and WFI or sleep on exit. To save further power debug system can be turned off.

2.6.1.3 Logic Cell Enhancements

Ultra Low Leakage logical cell library developed by ARM support special state retention cells that holds information in case of system power failure.

2.6.1.4 High density of code

For 16 bit instructions M0 has very high density of code. Hence application can be accommodated into small memory.

2.6.1.5 Low interrupt latency

M0 can handle interrupt in 16 clock cycles. This involves stacking of registers in stack, so that ISR can work without any software overhead.

NVIC can handle interrupt prioritization and ISR starting address so that exact IRQ can be serviced. Interrupt response is much lower than 8 or 16 bit system when supported by good programming practices.

2.6.1.6 User friendly

M0 is very user friendly as most of the software is in C language. This helps in shorter code development and easy portability.

2.7 Applications:

Mo offer high performance, low power and user friendly. Cortex-M series processors are carefully chosen from most of the microprocessors products. They are widely used in:

- Sensor nodes.
- Wireless communication chipset.
- Mixed signal application-specific standard product (ASSPs) and ASICs.
- In complex application systems as controller in subsystems.
- Security sensitive products as secure core processors e.g., SC000.which are used in SIM cards, electronic ID cards and banking/payment systems.

2.7.1 Advantages

Cortex Mo processors has number of key advantages

- Flexible interrupt management is provided using NVIC (Network Interrupt Controller).
- OS support features.
- Low power support like sleep modes.
- High code density.
- Integrated debugging.
- User friendly.
- High energy efficiency because of small size and better performance.
- For power management and boot sequence cortex ARM processor can be used as System Control Processor (SCP).

Chapter 3: Hardware and Software Details

In hardware Spartan 3E starter kit is used with Xilinx FPGA XC3S1600E which is compatible with Xilinx FPGA S3E500-4.

In software Xilinx ISE, KEIL embedded core and HyperTerminal/Teraterm soft wares are used.

In order to implement our task we need FPGA with over 12k gates. After investigation and comparison with others options Spartan 3E board is selected.

3.1 FPGA Spartan-3E as proto Board

Spartan 3E starter FPGA development board is a digital platform for various embedded implementation. It supports 16M bytes of SDRAM and 16M bytes of ROM. It has 50MHz crystal oscillator and base for secondary oscillator. USB2 power socket empowers all circuits, programming and data transfer modules. Also some other sub modules such as LCD screen, LEDs and switches, etc. LEDs can be used as event indicators. You can take advantage of Xilinx impact software such as Chipscope Pro, xmd, etc. In our case, this has helped to program and see the status of it easily.[10]

The Spartan 3E starter kit is compatible with Xilinx FPGA S3E500-4. In total, 500 logic gates, 20 hardware multipliers, 10,500 logic cells, 73 Kbits of DRAM, 360Kbits of RAM, having 4 clock sources and clock frequency of 300MHz are maintained.



Figure 3.1: Xilinx Spartan FPGA Board

3.2 Hardware overview

In order to support design Xilinx board provides many interfaces. Some are as under:

3.2.1 Slide switches

Four slides switches are here as shown in the figure 3.2. Located on lower right corner are designated as SW3, SW2, SW1 and SW0. When moved to up position a switch is connected to FPGA pin to high logic (3.3V). However, in down position will ground the switch. Switch does not have any active de bouncing circuit so it should be added by the programmer.[10]

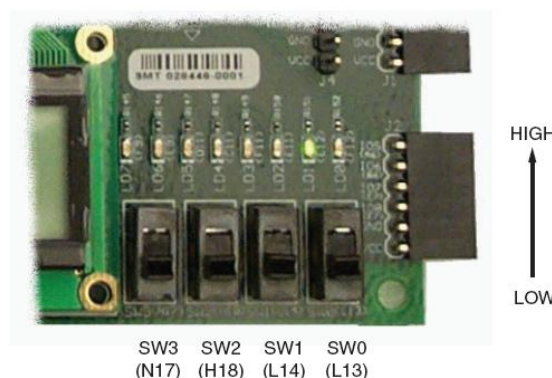


Figure 3.2: Four Slide Switches

UCF for slide switches is shown in the figure 3.3


```

NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTTL | PULLUP ;
NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTTL | PULLUP ;
NET "SW<2>" LOC = "H18" | IOSTANDARD = LVTTTL | PULLUP ;
NET "SW<3>" LOC = "N17" | IOSTANDARD = LVTTTL | PULLUP ;

```

Figure 3.3: UCF for Slide Switches

3.2.2 Push Buttons

Four push buttons are shown in figure 3.4. Location in the lower left corner, labeled as Btn North, Btn_East , Btn_south and Btn_west.

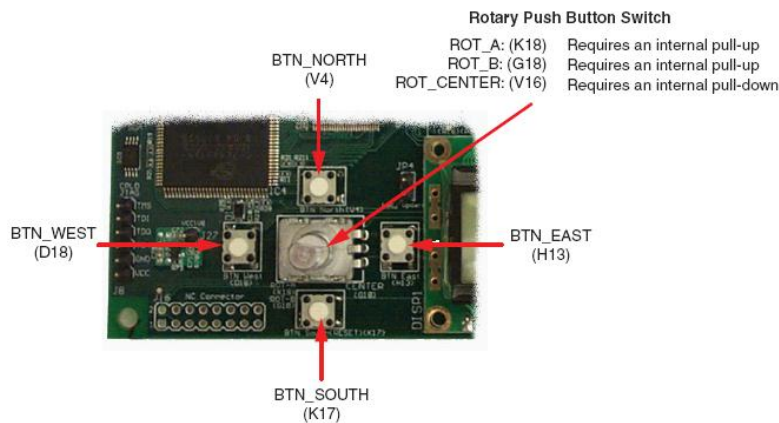


Figure 3-4: Rotary push buttons and four push buttons

In order to connected FPGA pin to high logic, push button needs to be pressed. The circuit with pull down resistor is shown in the figure 3.5.

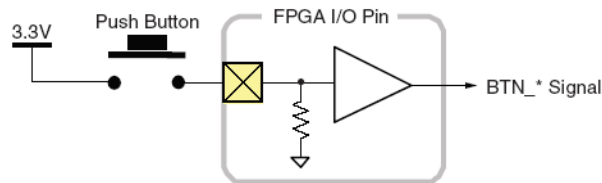


Figure 3.5: Push buttons must have internal pull down resistor in FPGA

In UCF user must define the pull down settings along with I/O standards and I/O pin and as shown in the figure 3.6

```

NET "BTN_EAST" LOC = "H13" | IOSTANDARD = LVTTTL | PULLDOWN ;
NET "BTN_NORTH" LOC = "V4" | IOSTANDARD = LVTTTL | PULLDOWN ;
NET "BTN_SOUTH" LOC = "K17" | IOSTANDARD = LVTTTL | PULLDOWN ;
NET "BTN_WEST" LOC = "D18" | IOSTANDARD = LVTTTL | PULLDOWN ;

```

Figure 3.6: UCF for Push-Button Switches

3.2.3 Rotary_Push Button

Its located in the middle of push buttons switches with three output. These are Rot_A, Rot_B and Rot_center. Rotary switches can acts as dual functions, when shaft turns output values changes and when pressed acts as a push button switch as shown in the figure 3.7.

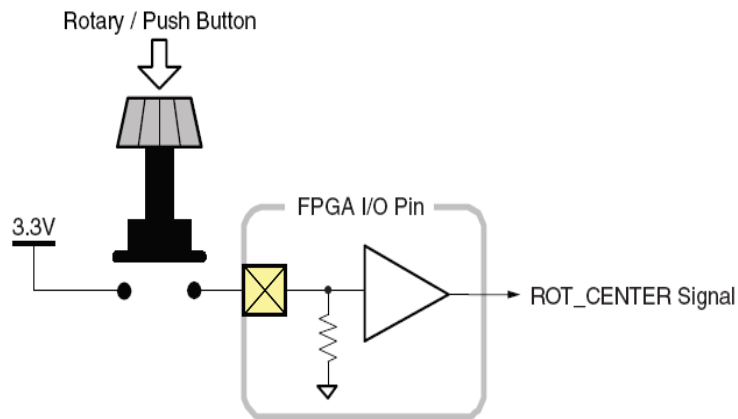


Figure 3.7: Pull up resistor for push button switch in FPGA

It acts like two push buttons connected by central shaft. These can act as make before break. When shaft is stationary in these positions both switches are closed. UCF constraints for four push buttons is shown in figure 3.8

```

NET "ROT_A"      LOC = "K18" | IOSTANDARD = LVTTTL | PULLUP ;
NET "ROT_B"      LOC = "G18" | IOSTANDARD = LVTTTL | PULLUP ;
NET "ROT_CENTER" LOC = "V16" | IOSTANDARD = LVTTTL | PULLDOWN ;

```

Figure 3.8: UCF file for Rotary Push-Button Switch

3.2.4 LEDs

Demo board is equipped with eight surface mount LEDs as shown in the figure 3.9. LEDs are labeled from LED0 to LED7.

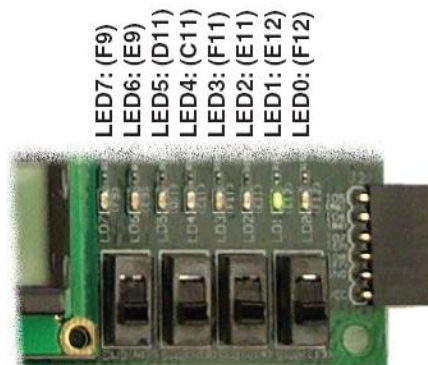


Figure 3.9: Eight surface mount LEDs

LEDs are grounded from one side via current limiting resistors of 390Ω. To drive LED, FPGA pin must be high. UCF of LEDs is shown in the figure 3.10

```

NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;

```

Figure 3.10: UCFs for Eight Discrete LEDs

3.3 Clock

Board supports three clocks as a source to FPGA as shown in the figure 3.11.

- Clock oscillator of 50 MHz.
- SMA connector for external clock input. It is high speed connector.
- 8 pin socket for oscillator

Clocks are connected to Bank 0 of FPGA. Each pin is also connected to DCM. The settings are as shown in table 3.1

Clock Input	FPGA Pin	Global Buffer	Associated DCM
CLK_50MHZ	C9	GCLK10	DCM_X0Y1
CLK_AUX	B8	GCLK8	DCM_X0Y1
CLK_SMA	A10	GCLK7	DCM_X1Y1

Table 3.1 Clocks, Global buffers and DCM

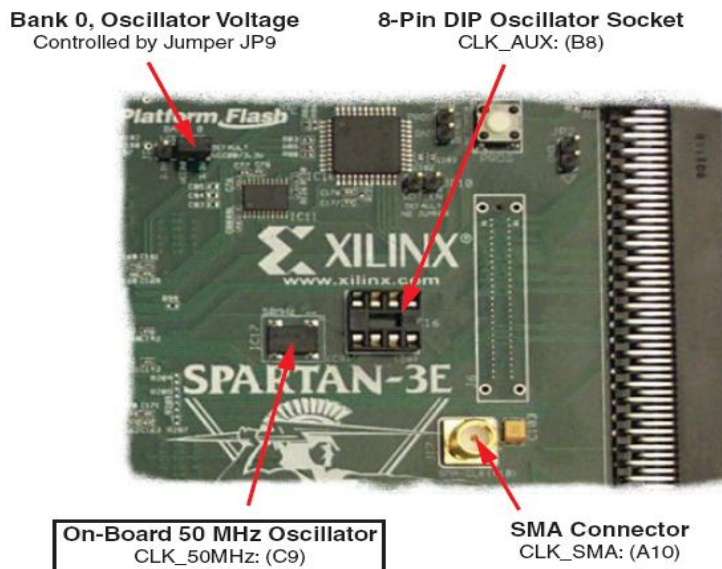


Figure 3.11: Available Clock

3.3.1 50 MHz oscillator

Onboard oscillator frequency is about 50 MHz with 40% to 50% duty cycle. It is accurate to ± 2500 Hz.

3.3.2 Oscillator socket

There is an eight pin socket for oscillator. It is normally used when operating frequency is greater than 50MHz. Also we can use DCM for other frequency generation.

3.3.3 SMA clock

SMA connector is used for external clock provision. For external device it is single ended clock.

UCF files for clock generation is shown in figure 3.12

```
NET "CLK_50MHZ" LOC = "C9" | IOSTANDARD = LVCMOS33 ;  
NET "CLK_SMA"   LOC = "A10" | IOSTANDARD = LVCMOS33 ;  
NET "CLK_AUX"   LOC = "B8"  | IOSTANDARD = LVCMOS33 ;
```

Figure 3.12: UCF for Clock Sources

3.4 FPGA Configuration Options

Board support many configuration options

- Code downloading via JTAG or on USB. This combination also program board Flash PROM and XC2C64A CPLD.
- Code downloading to 4M bit platform PROM, FPGA can be programmed from image stored in it using master serial
- Code downloading to 16M bit PROM; FPGA can be programmed from image stored in it using SPI.
- Code downloading to 128M bit flash, FPGA can be programmed from image stored in it using BPI UP or BPI down.

Figure 3.13 shows above described options in detail

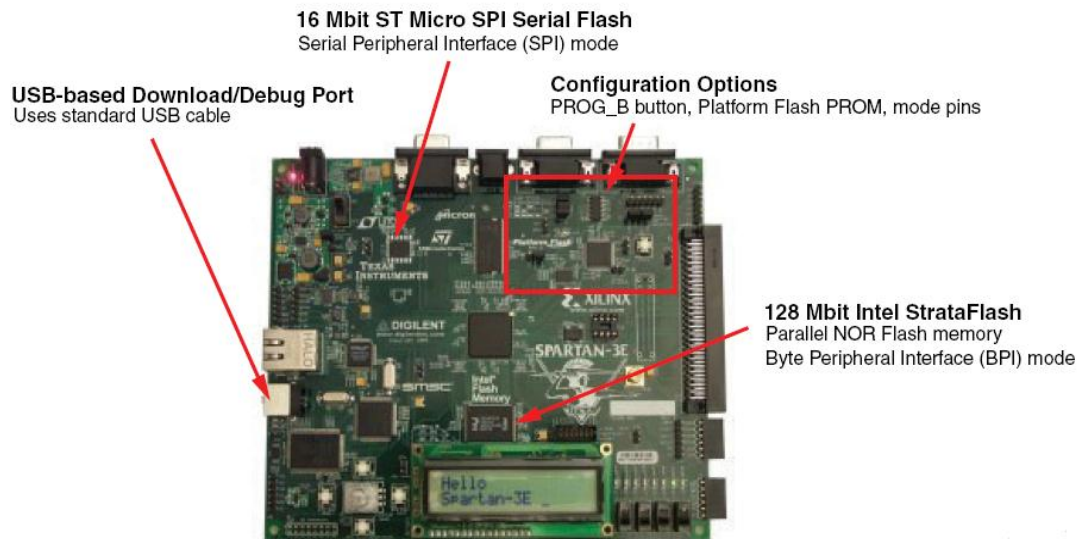


Figure 3.13: Configuration settings of Spartan-3E Starter Kit for FPGA

3.4.1 Programming push button

This button is shown in figure 3.14 that forcefully reconfigure fpga from selected configuration memory source. This can be done by using press and release of button.

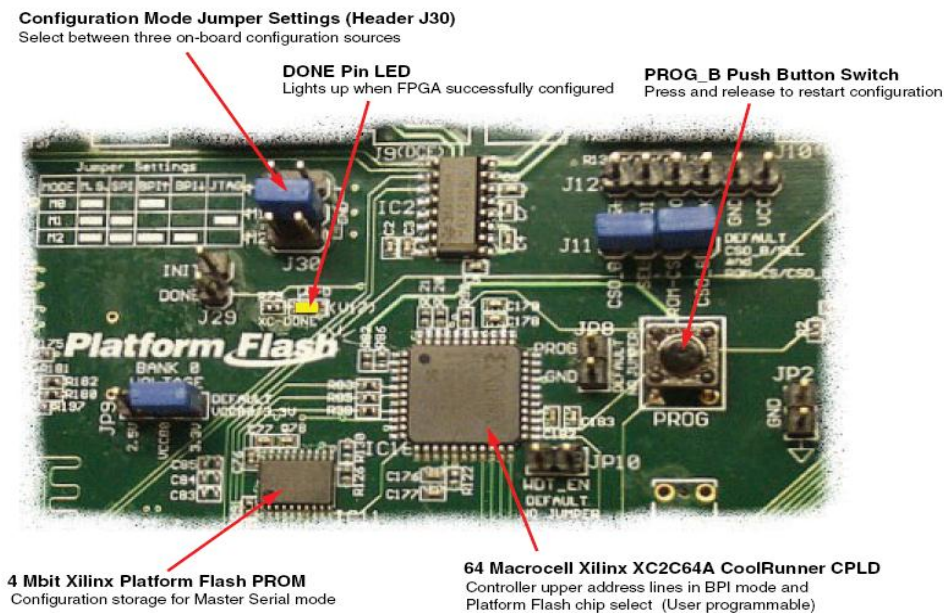


Figure 3.14: Detailed Configuration Options

3.5 Serial ports

There are two serial ports, DB9 female connector and DTE male connector. DCE connects directly to PC via standard serial cable. DTE connector connects madams, printers etc. Both connectors are shown in figure 3.15

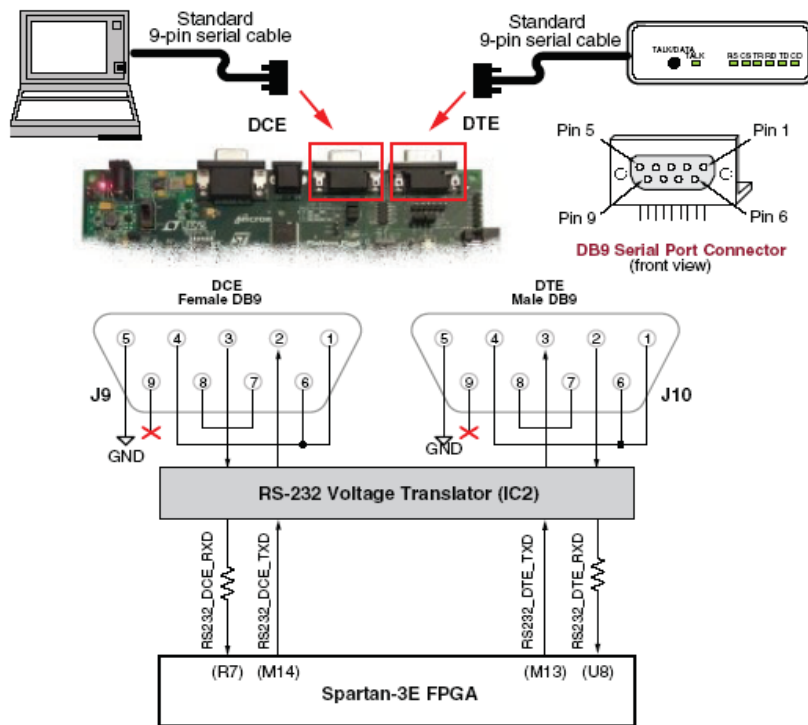


Figure 3.15: RS-232 Serial Ports

FPGA connects between two DB9 connectors. Its output data on LVTTTL or LVCMOS levels which converts the RS-232 voltage level. Figure 3.16 shows UCF for DTE and DCE ports.

```
NET "RS232_DTE_RXD" LOC = "U8" | IOSTANDARD = LVTTTL ;
NET "RS232_DTE_TXD" LOC = "M13" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = SLOW ;
```

UCF Location Constraints for DTE RS-232 Serial Port

```
NET "RS232_DCE_RXD" LOC = "R7" | IOSTANDARD = LVTTTL ;
NET "RS232_DCE_TXD" LOC = "M14" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = SLOW ;
```

Figure 3.16: UCF for DCE RS-232 Serial Port

3.6 FPGA programming

FPGA can be programmed using USB provided with kit. Attach USB and iMPACT programming software can be used directly to program FPGA. Programming options for parallel or serial PROM is not supported. As USB is connected a green LED turns on shows good connection. iMPACT can be directly launched from ISE project navigator as shown in the figure 3.17

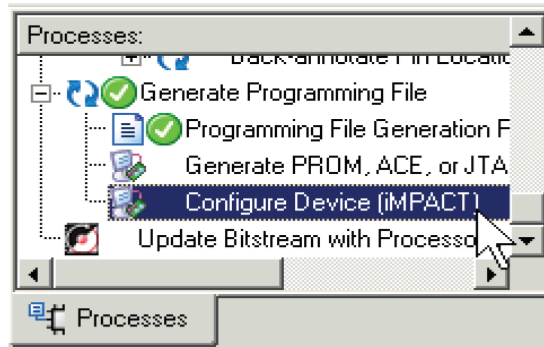


Figure 3.17: iMPACT open up option

As board is connected software recognizes three devices in the chain. To select FPGA, right click on it and assign new configuration file to it. Select file to program the device as show in the figure 3.18

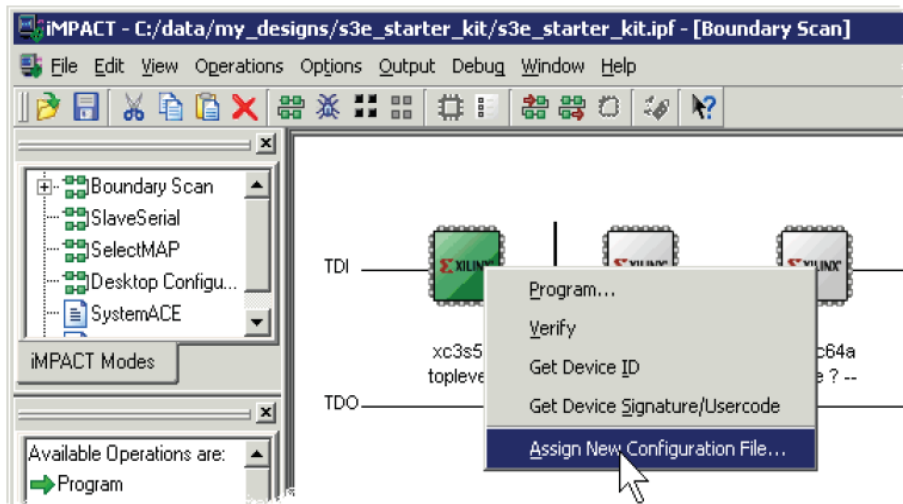


Figure 3.18: New configuration file for FPGA

When programming successful, FPGA application starts execution and DONE pin LED glows. The screens as shown appears in the figure 3.19

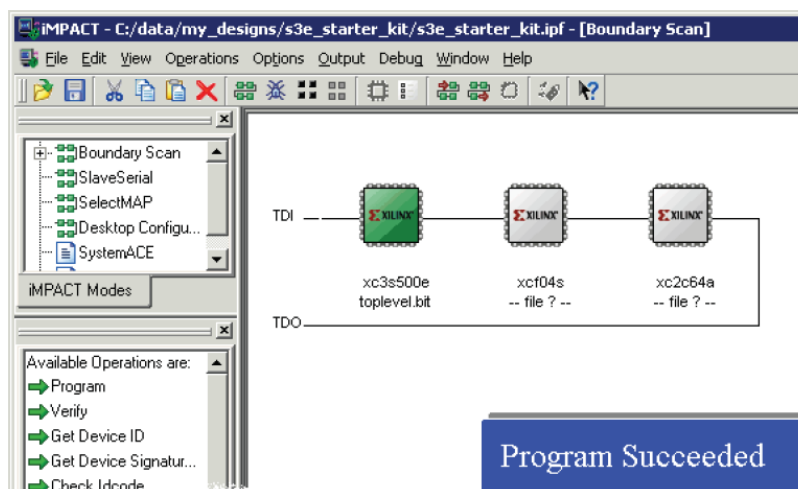


Figure 3.19: iMPACT Programming Succeeded

3.6.1 Generating Bit stream file

Bit stream is used for PROM programming. FPGA provides external clock to load external PROM. Oscillator starts from lower settings 1.5 MHz. PROM support higher frequency so this frequency can be increased. XCF04S flash support 25MHz.

Right click Generator programming file as shown in the figure 3.20

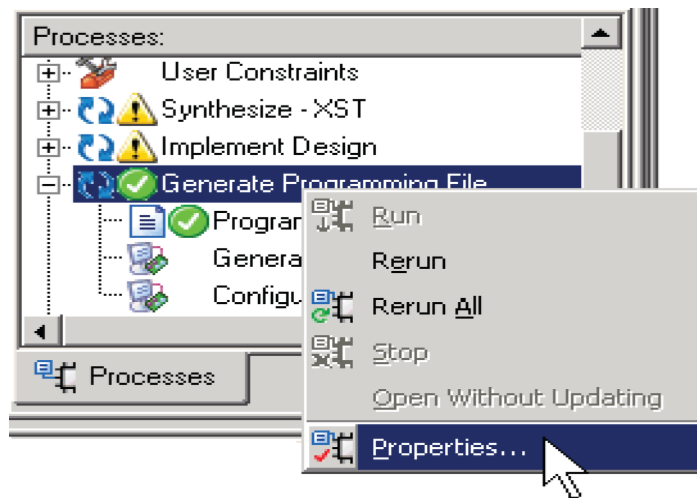


Figure 3.20: Set Properties for Bit stream Generator

In properties option increase clock to 25MHz. click on generate programming file. When file is generated, use option as shown in the figure 3.21

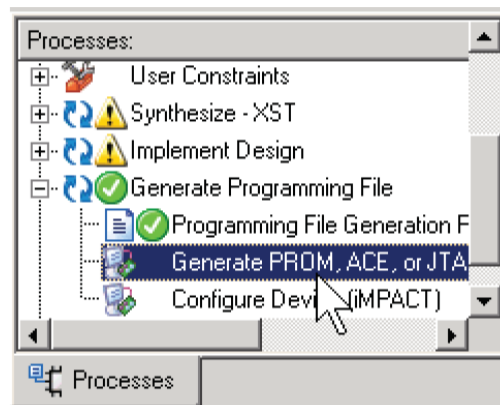


Figure 3.21: PROM, ACE, or JTAG File generation

As iMPACT on PROM file formatter show in the figure 3.22

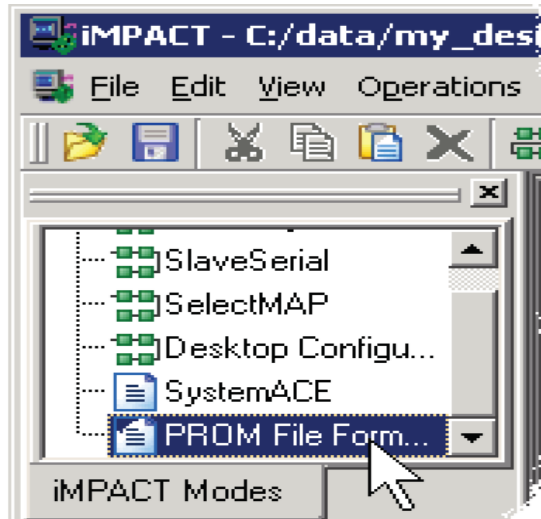


Figure 3.22: Click PROM File Formatter

Here target is Xilinx PROM. Select from any format. MCS (Intel Hex format) is popular. Enter path for file storage. Select XCF04S PROM from the options to program it Kit has Platform Flash PROM, XCF04S. Select XCF04s, list as shown in the figure 3.23

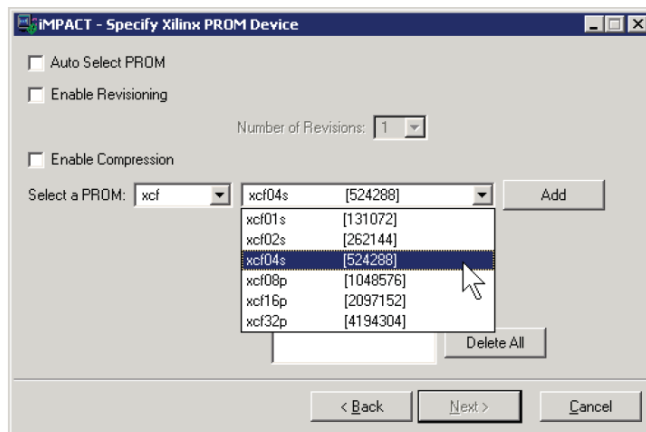


Figure 3.23: Selection of XCF04S Platform Flash PROM

Next step is to format the PROM and after that select bit stream file. Presses continue. PROM file thus created is shown in the figure 3.24

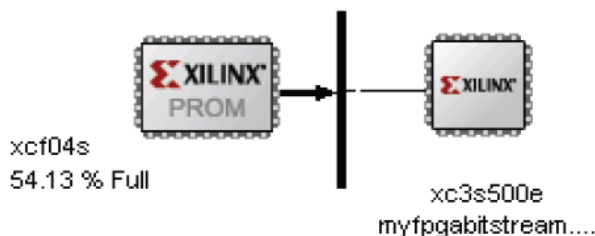


Figure 3.24: PROM File Formatter Succeeded

Chapter 4: Proposed Methodology

As already discussed in Figure 2.7 that M0 processor is connected with others modules through AHB-Lite bus (Advanced high speed bus) and APB (Advanced Peripherals Bus). So, in order to make M0 core we need files in which settings and configuration of M0 are included. Also we need files containing information about AHB and APB bus, clock settings modules. These are shown in figure 4.1. We are going to discuss this one by one.

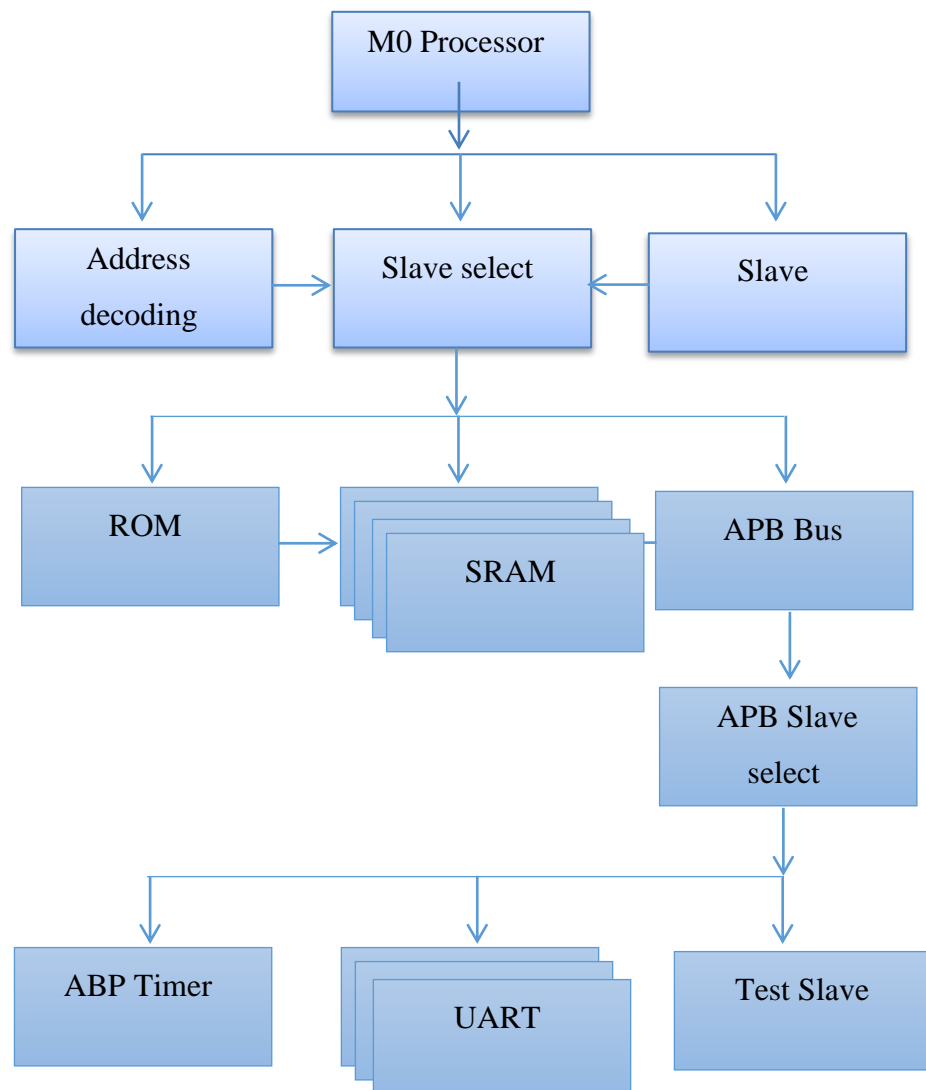


Figure 4.1 Block diagram of M0

4.1 M0 processor

In project we have use files like 'cmsdk_ahb_memory_models_defs.v'. This file defines memory models that we are using for our M0 core. In total M0 support following memory models.

- Using behavioral SRAM, Behavioral ROM model, with write disabled.
- AHB SRAM interface module with SRAM model , suitable for permitting read and write operations and FPGA flow.
- 32-bit flash memory with Flash wrapper.
- 16-bit flash memory with Flash wrapper.

In project we have used 'cmsdk_apb_dualtimers_defs.v'. In this files 32 bit down timers used in M0 core are initialized. In 'cmsdk_apb_watchdog_defs.v' file watchdog for M0 is initialized. In 'cmsdk_mcu_defs.v file M0 core main definitions are initialized.

4.2 Slave

Slave is defined in 'cmsdk_ahb_default_slave.v' file. Slave responds to transfer in case master bus tries to access undefined address. When bus is idle a zero wait state or ok response is generated, however, slave generates error response when sequential or non-sequential transfer takes place. AHB default slave components are shown in figure 4.2

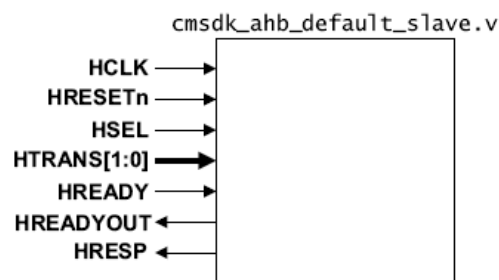


Figure 4.2: 'cmsdk_ahb_default_slave.v' model

4.3 Slave select

Slave select module is named as 'cmsdk_ahb_slave_mux.v' file. It supports upto 10 slaves that are connected with AHB bus. Parameters that define slave port usage are also defined in it, so that additional logic may not be generated by synthesis. Block is shown in the figure 4.3

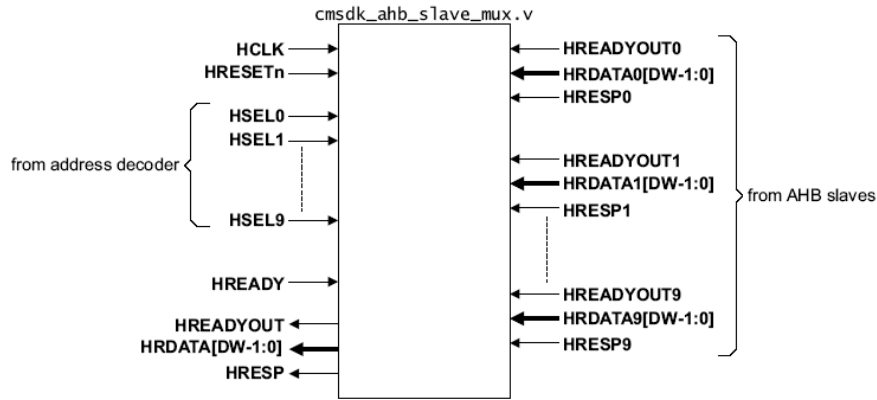


Figure 4.3: 'cmsdk_ahb_slave_mux.v' model

Address decoder determine selected slave and generates correspondence HSEL signal to AHB slave and AHB slave multiplexer. Slave use register version of HSEL as data and signal are valid during data phase. For more than 10 slaves more AHB slave modules can be cascaded.

4.4 Address decoding

In project 'cmsdk_mcu_addr_decode.v' file is used in conjunction with others file. The main aim of this file is to check the M0 generated address and compare it with assigned addresses in to the modules of M0. If valid address is matched it grants the AHB bus access to that module for data transfer.

4.5 APB Bus

APB bus platform is provided in 'cmsdk_apb_subsystem.v' file. It is used to interface module slike APB timers, Dual input timers, APB UART, AHB to ABP Bridge, Test slave and IRQ synchronizer. It includes signals to enable all its modules. When address generated by AHB bus matches the address allocated in APB Bus it generates corresponding enable signal. Here HCLK (Clock for AHB) is converted into PCLKG (Gated clock for APB bus). For our code we are using the same clock frequency for AHB to PCLKG. APB bus is used here as 32 bit data bus and 16 bit Address Width.

4.6 ROM

In project 'cmsdk_ahb_rom.v' file is used with other files. ROM is of 32 bit data bus and 16 Bit address bus and. The ROM model is shown as figure 4.4

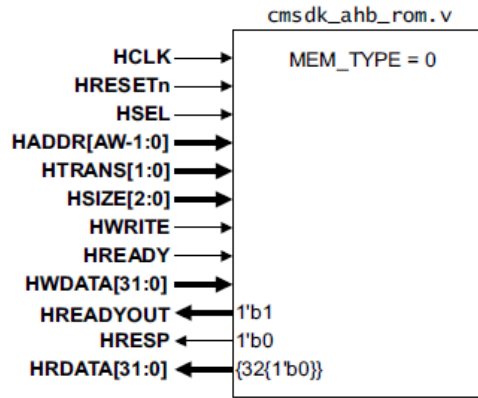


Figure 4.4 cmsdk_ahb_rom model

4.6 SRAM

Read write operations is performed with zero wait state. The design only supports 32 bit memory interfacing. Here, we have made four SRAMs of 8x2048 Bits. The data transferred in it byte wise. The main model is shown, in figure 4.5

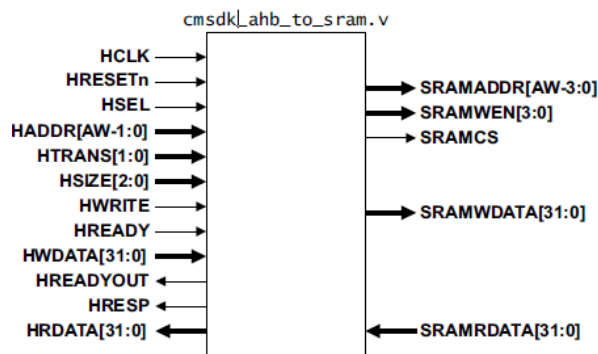


Figure 4.5 cmsdk_ahb_to_sram.v model

4.7 ABP Slave select

The slave multiplexer is included in 'cmsdk_apb_slave_mux.v'. It supports upto 16 slaves. In order to do this it use four bits and PADDR to generate corresponding PSEL signal. PADDR can be configured to decode slaves. Figure 4.6 shows APB slave mux module

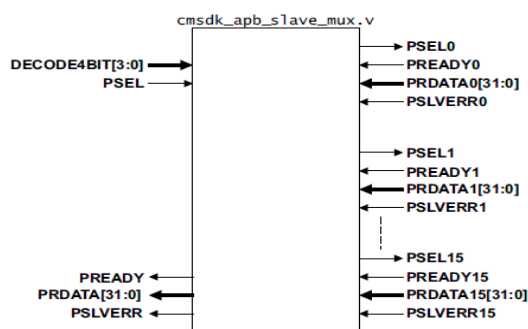


Figure 4.6 cmsdk_apb_slave_mux.v module

4.8 ABP Timer

Timer is included in project using `cmsdk_apb_timer.v` file. It is a 32 bit timer ,following features:

- Interrupt be generated using `TIMERINT` signal as counter reaches zero.
- `EXTIN` signal can be used as external input signal to enable timer.
- APB timer reaches zero and same time software clears, interrupt is status set to one ,previous interrupt status.

ABP timer block diagram is shown in figure 4.7

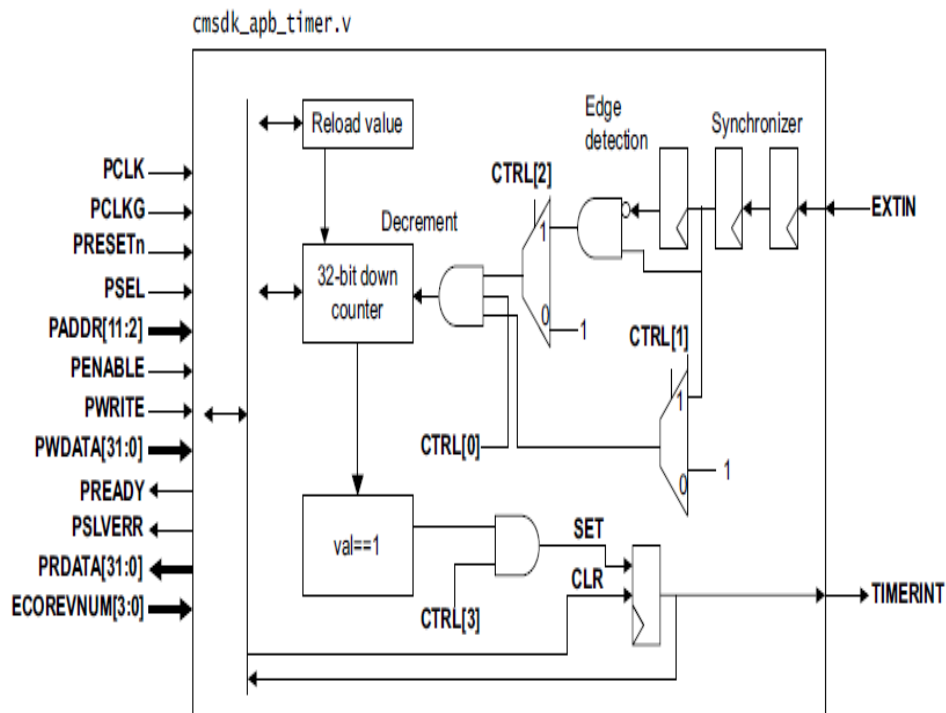


Figure 4.7 block diagram of timer

4.9 ABP UART

`cmsdk_apb_uart.v` is APB UART, supports 8 bit communication, simple in design,. It has no parity and zero as stop bit. Block diagram of UART is show in figure 4.8.

UART has two buffers one for data reception and another for data transmission. Interrupt handling execution time is short this leaves sufficient time for processor.

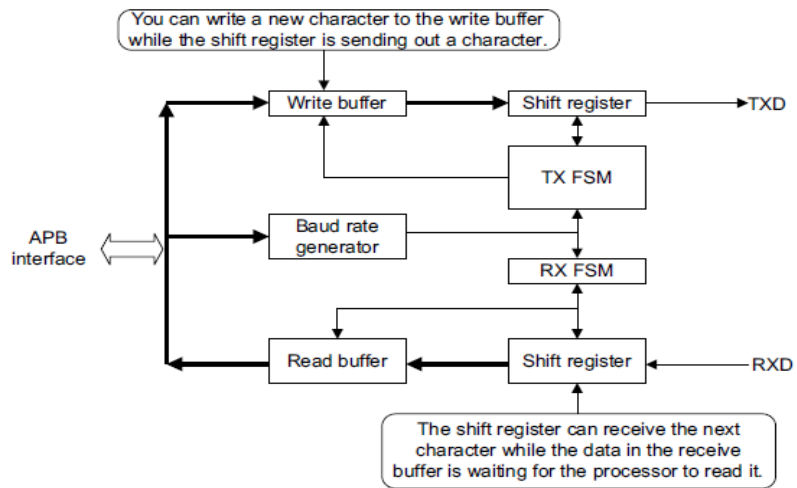


Figure 4.8 block diagram of ABP UAR

Chapter 5: UART communication

5.1 M0 processor

In embedded systems, processors and other integrated circuits are interlinked to make system functional. Communication protocols have to be followed to pass information between these circuits. There are many communication protocols used for data exchange. Major communication protocols are two i.e serial and parallel.

5.2 Serial vs. Parallel

In parallel transfer protocol multiple bits are transferred at same time. Data transfer occurs in form of 0's and 1's. Data is transmitted at buses of eight, sixteen and more wires.

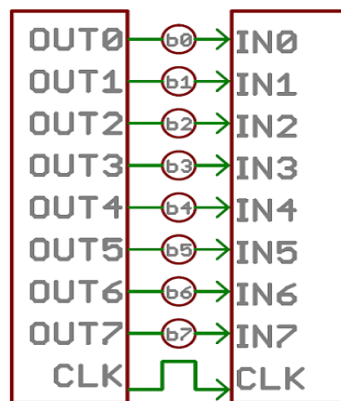


Figure 5.1 parallel communication

The above picture making it clear that in above parallel communication out of used 9 wires, 8 bit data bus is using 8 wires which are controlled by a clock. At every clock pulse a byte is transmitted. At a time a single bit is streamed in above serial interface.

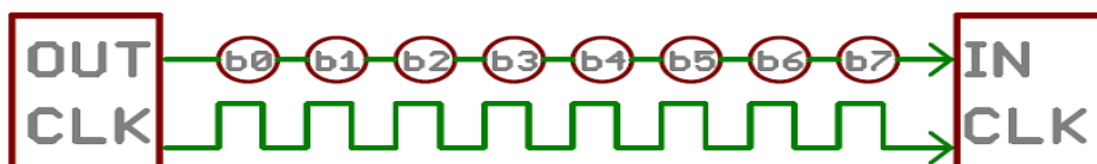


Figure 5.2 serial communication

In above serial interface is one bit at every clock pulse is transmitted. Only 2 wires are required in this case.

Over serial communication, parallel communication has also many advantages.

Following are its advantages, it is

- Relatively easy to implement
- fast
- and straight forward.

but more input/output lines are needed in parallel communication .As in most of the projects I/O lines of the microprocessor are often very few and precious.

5.3 Types of serial Interfaces

There are two types of serial interfaces

- 1. Asynchronous Serial Interface.**
- 2. Synchronous Serial Interface.**

5.3.1 Synchronous serial interface

In synchronous serial interface there are data lines and a clock signal. The clock signal is actually the common clock which all devices on the serial bus has to share. Serial data transfer is in this case is linked with clock.

Example: I²C and SPI.

5.3.2 Asynchronous serial interface

In Asynchronous serial interface for data transfer no external clock signal is required. This serial interface has advantage that it require minimum wire and I/O pins.

Its drawback is that we have to add some functionality for reliable data receiving and transferring data. In embedded electronics ,the asynchronous serial interface is widely used .For example if u have to add Bluetooth, XBee's, GPS module and other devices .

5.4 Rules of Serial

For error free and accurate ,robust data transfer In Asynchronous serial protocol following rules has to follow. These are as under

- Baud rate(data bits per second) of the data,
- Data bits of the data,

- Synchronization bits of the data, and
- Parity bits of the data,

There are many ways to send data serially. This protocol is configurable. For error free data communication same exact protocol must be used at both transmitting and receiving device on serial bus.

5.4.1 Baud Rate Settings

In serial communication at serial line baud rate is actually specifies how fast data is transmitted in a second. It actually defines the time in second that a single bit takes to transmit. its unit is bits/sec.

Commonly used baud rates are 1200, 2400, 4800,19200,38400,57600, and 115200.Where speed is not critical 9800 is the commonly used baud rate. It's essential for serial communication to establish that both devices operate at same rate i.e. at same baud rate.

For most of the embedded controller projects the highest speed is 115200, above that mostly error comes in transferred data. So, for successful data transmission appropriate baud rate has to set.

5.4.2 Format of data

In serial communication data is transmitted in form of packet or frame of bits. The frame is formed as shown in figure below. Synchronization bits start, stop and parity is added to data bits to form complete packet.



A serial frame. Some symbols in the frame have configurable bit sizes.

Figure 5.3 data framing

5.4.3 Chuck of Data

In serial packet the data chunk is actually the true data in frame. We call it as chunk as its size is not specifically stated. The standard size is 8 bit means 1 byte, it can be set to anything from 5 to 9 bits. A 7 bit data size is more efficient than 8, if we are transferring ASCII characters.

After a data part in packet, endianness of the data has to check also.by default LSB(least significant bit) format is used generally to transfer data.

5.4.4 Synchronization bits

Synchronization bits are the start and stop bits, these are actually the special bits transferred with each chunk of data. These bits actually indicate start and end of the packet. Start bit is 1 always, whereas the stop bit can be one/two. Commonly 1 stop bit is used.

These synchronization start bit actually indicate idle data line that goes from 1 to 0, and stop bit(s) is changed back to the idle state means it hold the line at 1.

5.4.5 Parity bits

Error checking at low level is done by parity bit in serial communication. It has two types: odd or even parity.

It is actually calculated by adding all 5-9 bits of data byte and if the sum is even the bit is set or not.

As an example, even parity is set and is added to data byte, 0b10011101, which has an odd number of 1's (5), the parity bit become now set to 1. Conversely, if the parity is set to odd, the parity bit be 0 in that case.

To set parity is optional and not used mostly. It can help to transmit data in noisy medium. Its drawback is that it slows down data transferring and at both sides error handling has to implement.

When it's detected by parity that transferred data is not accurate, that data must be resent.

Serial Settings: 9600 8N1 (an example)

Serial settings of 9600 8N1 means - 9600 baud, 8 data bits, no parity, and 1 stop bit .

When a device transmit ASCII character 'K' and 'O'. It creates two packets of binary value "01000001" and "01011010".synchronization bit is then appended. Least significant bit is sent first.



Figure 5.4 data packet for serial protocol

Data is transferred at 9600 baud rate i.e. per bit 104 μ s.

When we transmit every bit one start bit and one stop bit, and 8data bits have to transmit.so 9600 baud rate means 9600 bits per second is sending.

5.5 Wiring and Hardware

Two wires are here, one is for sending data and other for receiving data. Devices connected with serial should have two serial pins: RX and TX.

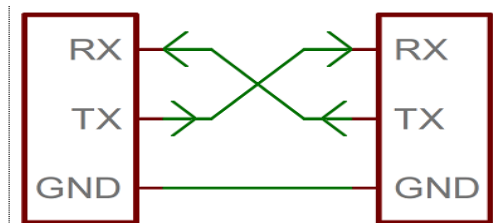


Figure 5.5 Hardware Connection

Serial data can be transmitted in two ways

- Full duplex.
- Half duplex.

In full duplex communication both devices can receive and send simultaneously. Whereas in Half-duplex communication serial devices sending and receiving take turns.

5.5.1 Hardware Implementation

At signal level popular hardware implementations of serial are TTL and RS232.

TTL serial signal of microcontroller voltages supply range is usually 0V to 3.3V or 5V. bit value 1 actually represent an idle line and bit value 0 represent a start bit or a data bit of value 0 or 0volt GND signal.

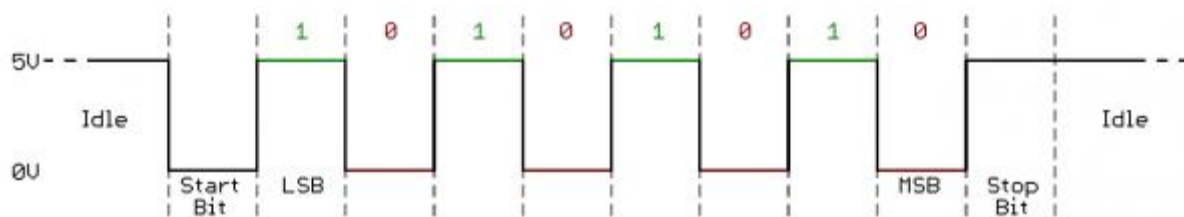


Figure 5.6 5 volt TTL signal

When the TTL signals are flipped on its heads, in old communication the protocol is RS232. RS232 signals usually range between -13 to +13 volts. Low voltage (-5V, -13V) are a stop bit, data bit, the idle line whereas high voltages means a start bit or 0 value data bit.

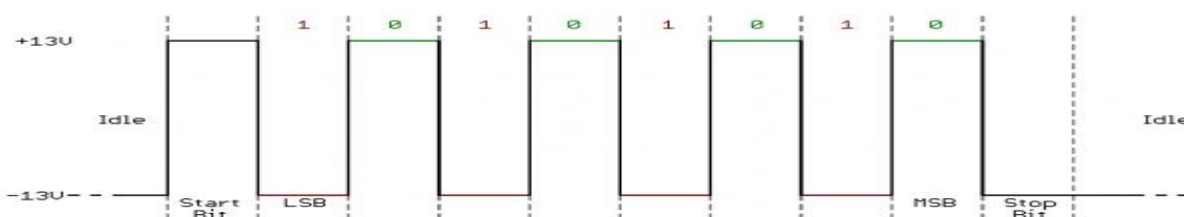


Figure 5.7 13 volt TTL signal

In embedded circuits TTL is much easier to implement. But in long transmission lines, low voltage levels are more susceptible to losses, there RS232 is suitable. More complex standard like RS-485 is more suitable.

TTL serial device and RS232 bus can't be connected directly. Shifting of these signals is mandatory and signals should match up for accurate serial communication.

UARTS are used to convert data on parallel bus to and from a serial interface.

5.6 UARTs (Universal Asynchronous Receiver Transmitter)

UART is used to create the serial packets and control physical hardware lines.

A universal asynchronous receiver/transmitter (UART) implements serial communication. UART is electronic block between parallel and serial interfaces. At one end of UART is a bus of eight or above data lines (plus some control pins) and on the other side is the two serial wires, RX and TX.

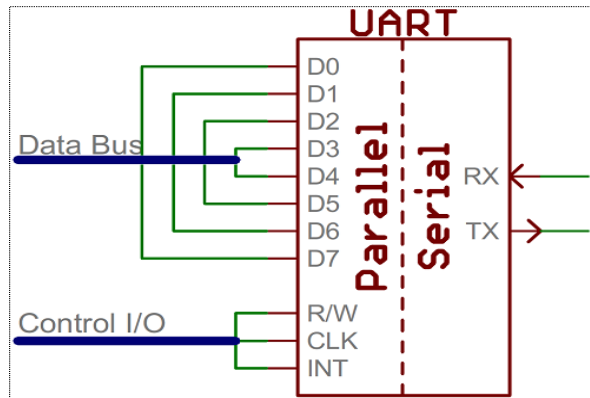


Figure 5.8 UART interface

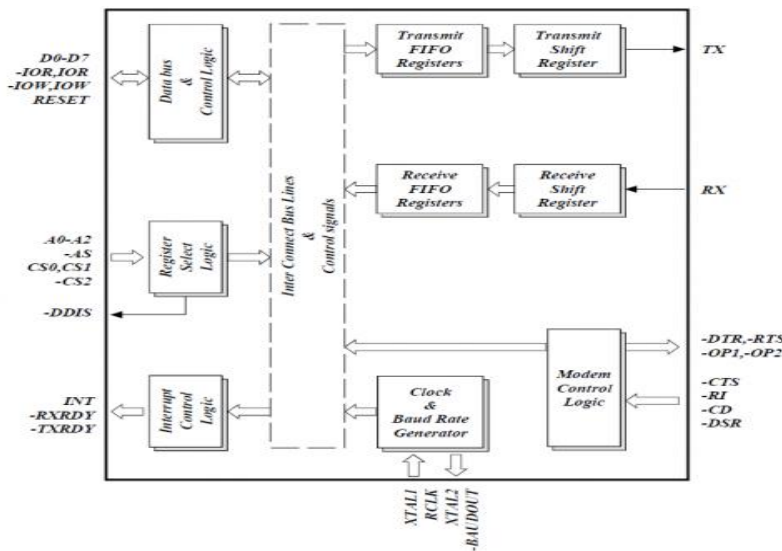
Simplified UART interface. Parallel on one side, serial on the other side.

UART ICs are found commonly inside microcontrollers, they also exist as standalone ICs. Some have none, some have one, and some have many UARTs. Datasheet will explore that the microcontroller has UART or not. For example, the Arduino Uno which is based on ATmega328 has single UART while the Arduino Mega which is built on an ATmega2560 has four UARTs.

Serial data is both transmitted and received via UART.

UART is used for sending and receiving data. At transmitting side, a data packet is created, synchronization and parity bits are appended and at TX line with precise timing and baud rate, packet is sent.

At receiving end, the RX line has to sample according to expected baud rate, synchronization bits are separated and data is extracted out.



Internal UART block diagram (courtesy of the Exar ST16C550 datasheet)

Figure 5.9 UART internal block diagram

Advanced UARTs saves data in a buffer and can be used by the microcontroller for processing. These buffers are in the form of first-in-first out (FIFO) basis and can be of few bits to thousands of bytes.

5.7 Software UARTs

Software serial is like bit banging process that is directly controlled by processor. For example Arduino libraries are software serial. Software serial working is processor intensive but not very precise as a UART but it works.

5.8 Common Pitfalls

Few mistakes an experienced engineer also can do. So, following things should be kept in mind while working on serial and for making serial hardware connections.

5.8.1 TX-to-RX, RX-to-TX

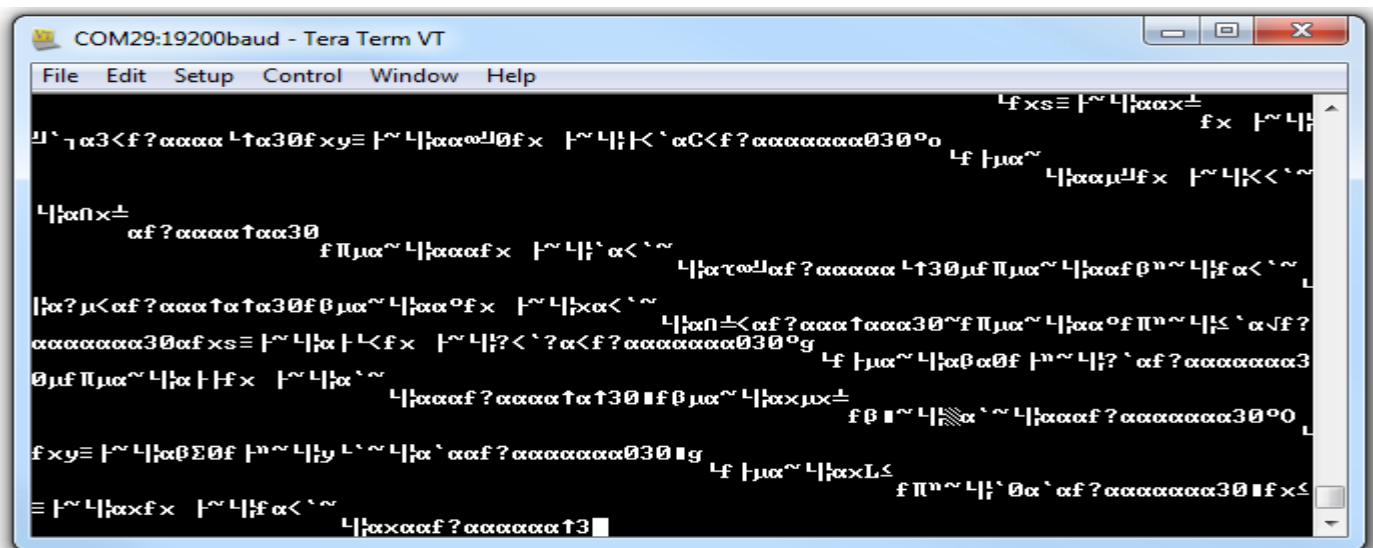
Most commonly committed mistake is often people forget to cross the RX and TX lines between serial devices.



Figure 5.10 Pro Mini design

5.8.2 Baud Rate matching

The language of serial communication is baud rate. To avoid missing or interpretation of data two devices should speak at the same speed. If there is garbage data on receive line, then make it sure that the baud rate is matched or not.



When baud rate of transmitted data is 9600 bps, but receiving data baud rate is 19200 bps. Baud rate mismatch results in garbage data.

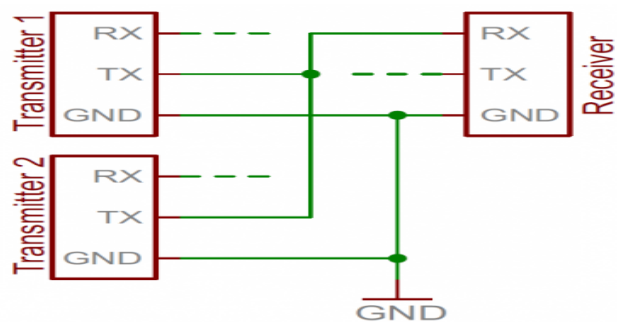
Figure 5.11 Mismatched data

5.8.3 Problem of Bus Contention

Bus contention is a problem which comes when more than one device try to transmit on single serial line. Serial communication is actually designed in such a way that those just two devices communicate at serial bus at a time.

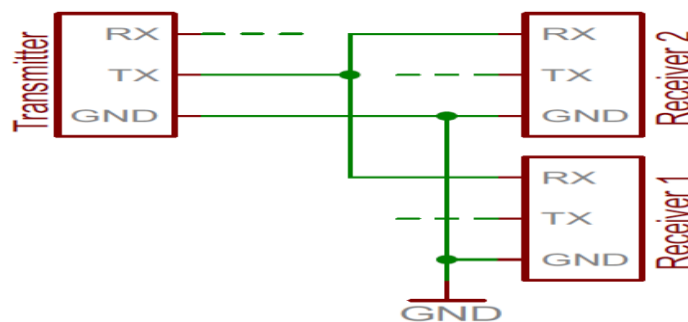
As an example, if a GPS module is connected to Arduino, then its TX line with RX line of Arduino board. But as TX pin of USB to serial converter is already connected to RX pin of the Arduino used to program the Arduino to watch Serial Monitor.

When both the FTDI chip and GPS module trying to transmit on the same line at same time there odd situation comes of contention.



The bus contention occurs when two transmitters send to single receivers.in that case neither of the devices can send data. In worst situation transmit lines of both devices clogs and that situation is rare and protection must be here to avoid that situation.

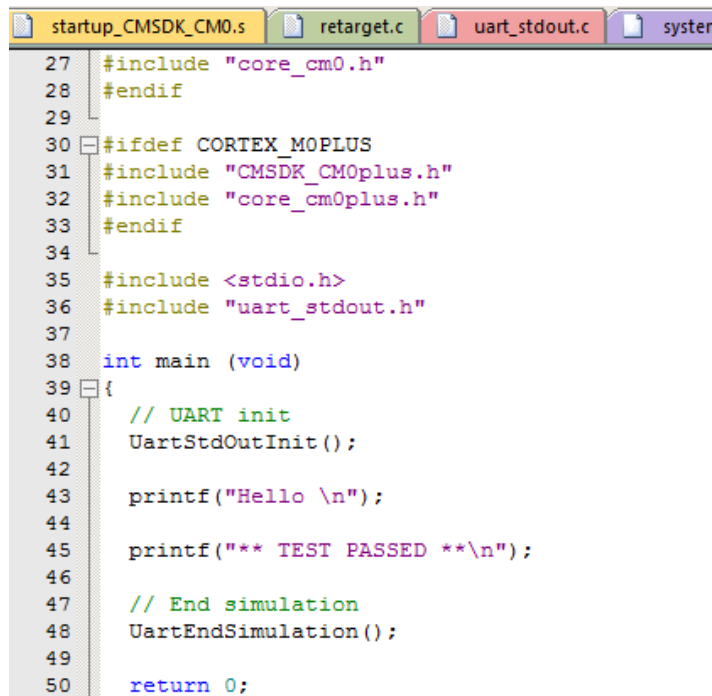
A single transmitting device can be connected with multiple receiving devices. For example, when an lcd has to connect with Arduino then we connect Arduino TX line with LCD's module RX line. Arduino's TX line is also connected with RX line of USB programmer's.in that scenario also there is one device in taking control of transmission line.



When TX line is distributed its also dangerous from a firmware point of view, as it's not clear which device achieving which transmission .LCD can get unknown data which can led it in an unknown state.

Chapter 6: EXPERIMENTS AND RESULTS

M0 core can be programmed in C language, so we need some tool to make hex file of the project. In order to carry out our task we use Keil integrated development environment for our project. In our program we are printing simple message as shown in the figure 6.1.



```
27 #include "core_cm0.h"
28 #endif
29
30 #ifdef CORTEX_M0PLUS
31 #include "CMSDK_CM0plus.h"
32 #include "core_cm0plus.h"
33 #endif
34
35 #include <stdio.h>
36 #include "uart_stdout.h"
37
38 int main (void)
39 {
40     // UART init
41     UartStdOutInit();
42
43     printf("Hello \n");
44
45     printf("** TEST PASSED **\n");
46
47     // End simulation
48     UartEndSimulation();
49
50     return 0;
```

Figure 6.1 Keil Program

The message is simple Hello ** TEST PASSED **. The final file is saved as 'image.hex'

6.1 Simulation

In order to simulate our C code, we now need ISE navigator. Open ISE and select the FPGA as shown in the figure 6.2

In file 'cmsdk_ahb_rom' file give the path of 'image.hex'. as show, in the figure 6.3

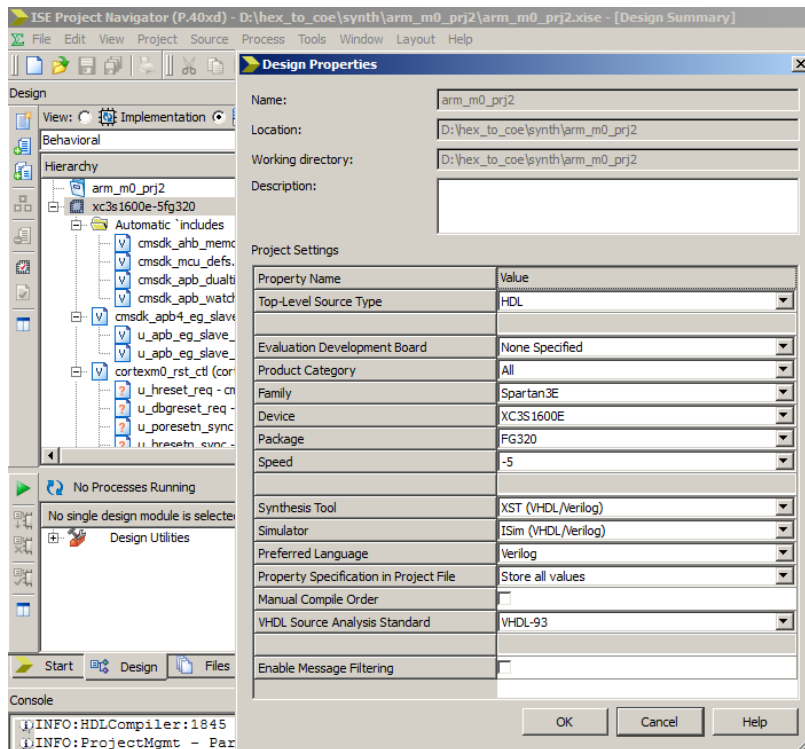


Figure 6.2 FPGA settings

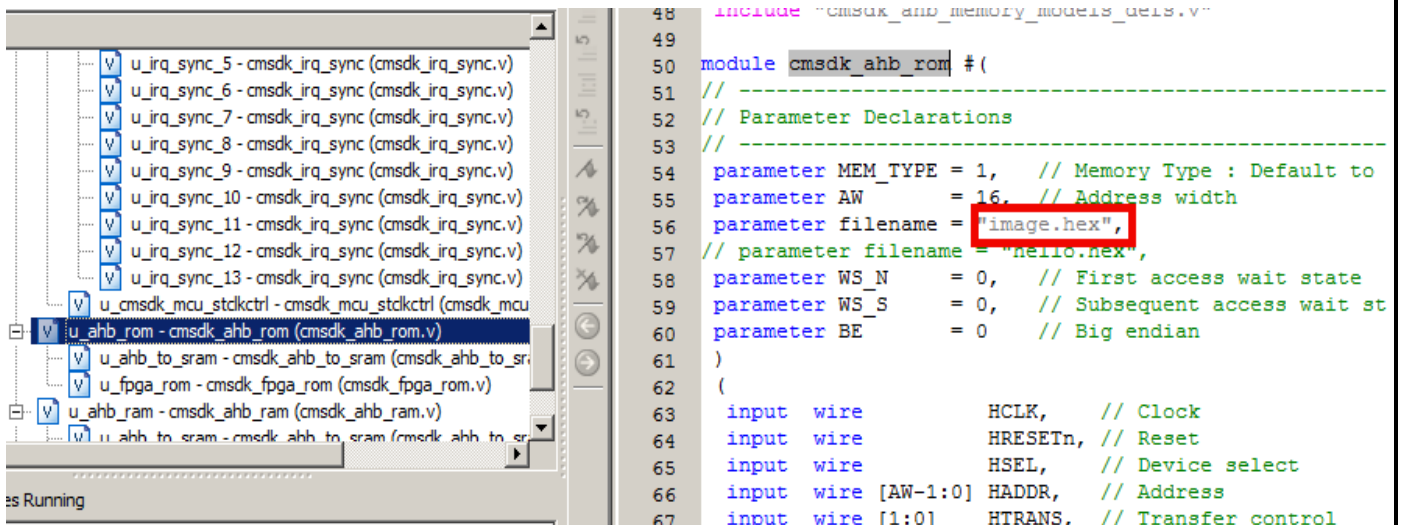


Figure 6.3: Path for image.hex file in cmsdk_ahb_rom.v

Also, add file 'image.hex' in 'cmsdk_fpga_rom' file as shown in the figure 6.4

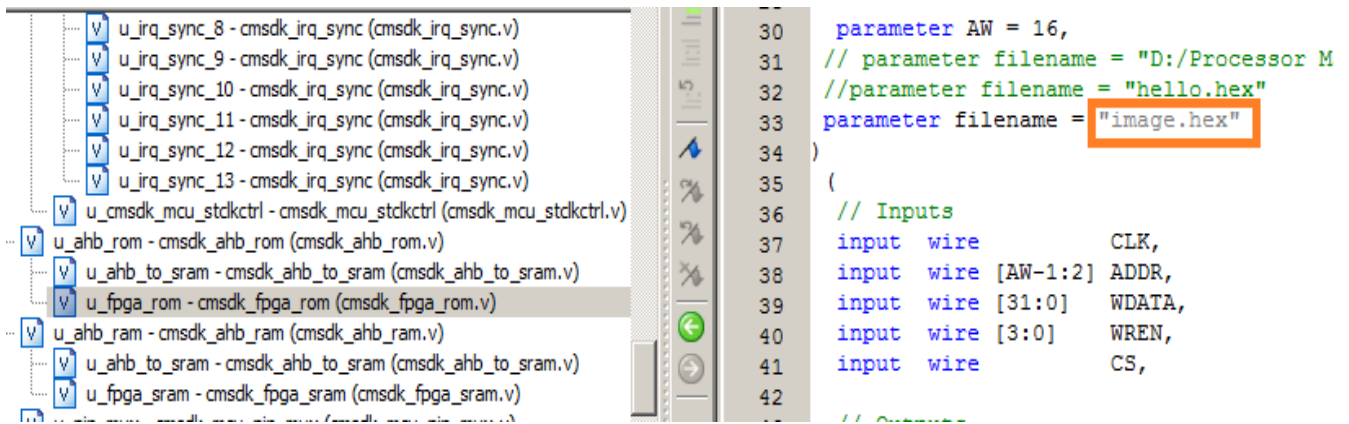


Figure 6.4: Path for image.hex file in cmsdk_fpga_rom.v.

In order to make M0 core we need four BRAMs each of 8x2048 byte. Make four files in same project directory. Save as "BRAM1.coe", "BRAM2.coe", "BRAM3.coe" and "BRAM4.coe". Add files in the cmsdk_fpga_rom.v file as shown in the figure 6.5

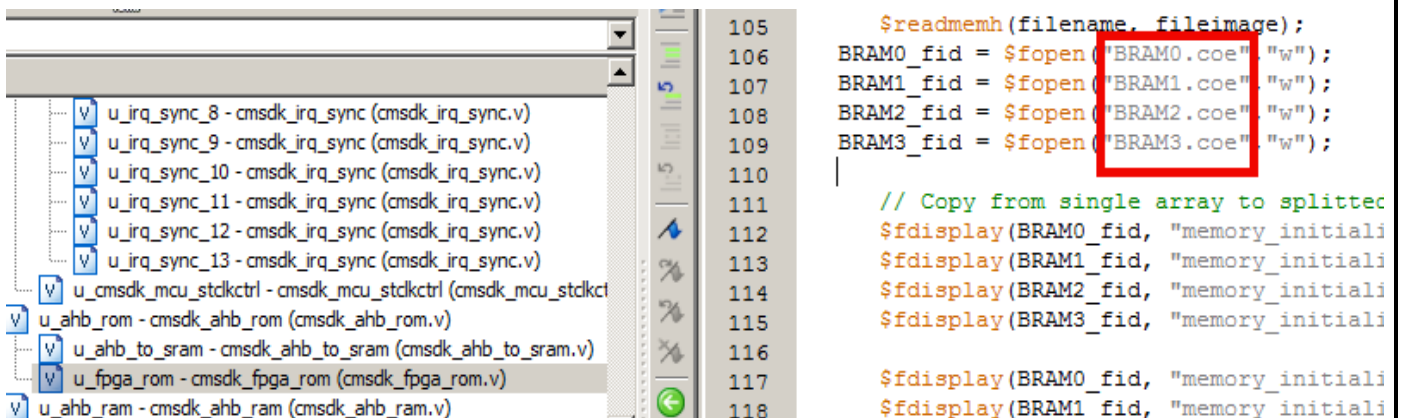


Figure 6.5: Path for BRAM files in cmsdk_fpga_rom.v.

After that simulate the code and output is shown in the figure 6.6.

The simulation results are confirmed by matching first byte of 'image.hex' file with "BRAM1.coe" file. Both bytes should be same. We can check this by modifying first byte of 'image.hex' file and simulate the M0 core. After simulation both the bytes should be same.

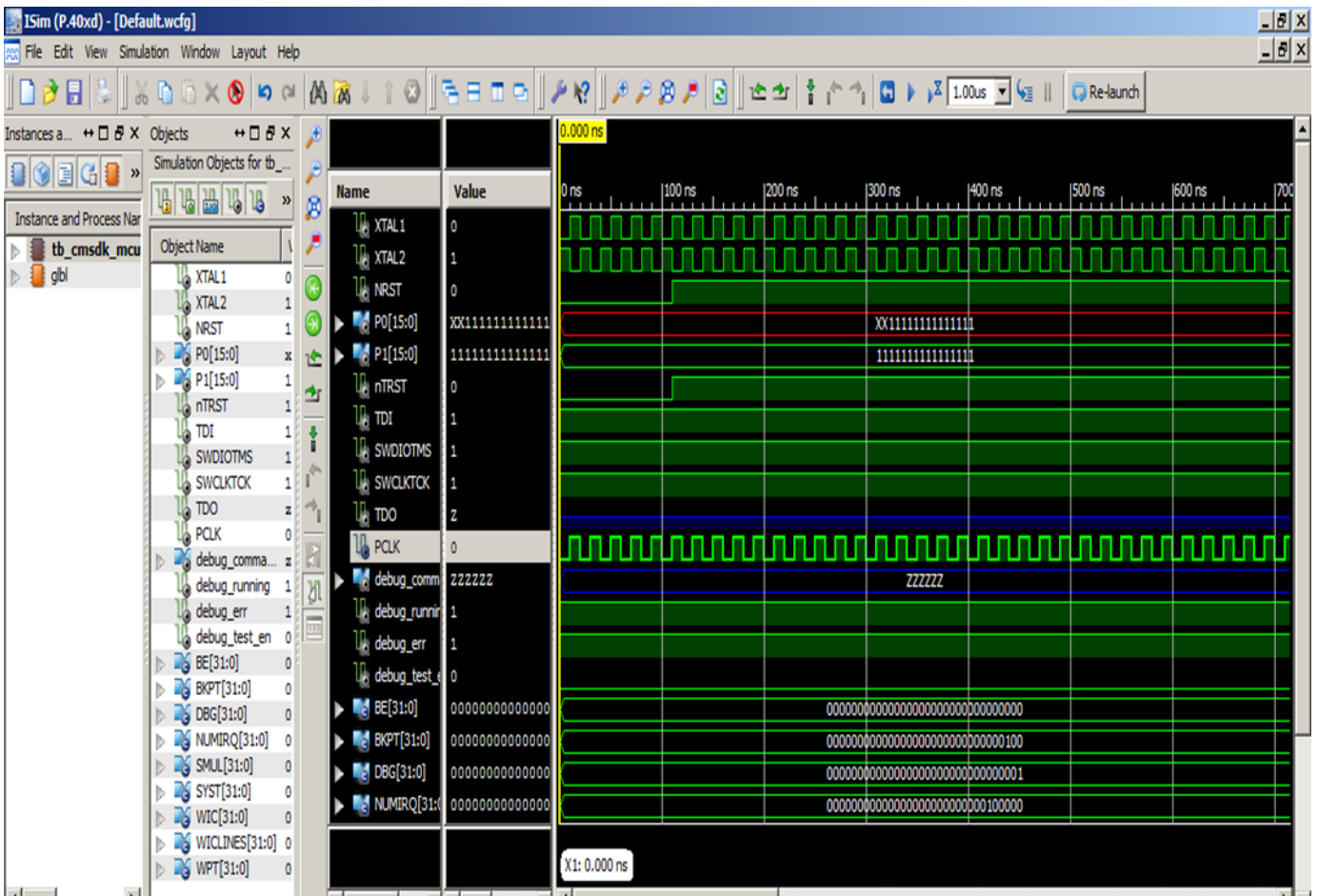


Figure 6.6: Simulation of M0 core processor

6.2 Implementation

In order to implement M0 core, so open project 'ARM_prj2_m0'. Add 'image.hex' file in 'cmsdk_ahb_rom' as shown in the figure 6.7

```

50 module cmsdk_ahb_rom #()
51 //
52 // Parameter Declarations
53 //
54 parameter MEM_TYPE = 1, // Memory Type : Default
55 parameter AW = 16, // Address width
56 parameter filename = "image.hex",
57 parameter WS_N = 0, // First access wait state
58 parameter WS_S = 0, // Subsequent access wait state
59 parameter BE = 0 // Big endian
60 )
61 (
62 input wire HCLK, // Clock
63 input wire HRESETn, // Reset
64 input wire HSEL, // Device select
65 input wire [AW-1:0] HADDR // Address

```

Figure 6.7: Path for image.hex file in cmsdk_ahb_rom.v.

Make all four BRAMs using IP core generator with specifications as shown in the figure 6.8

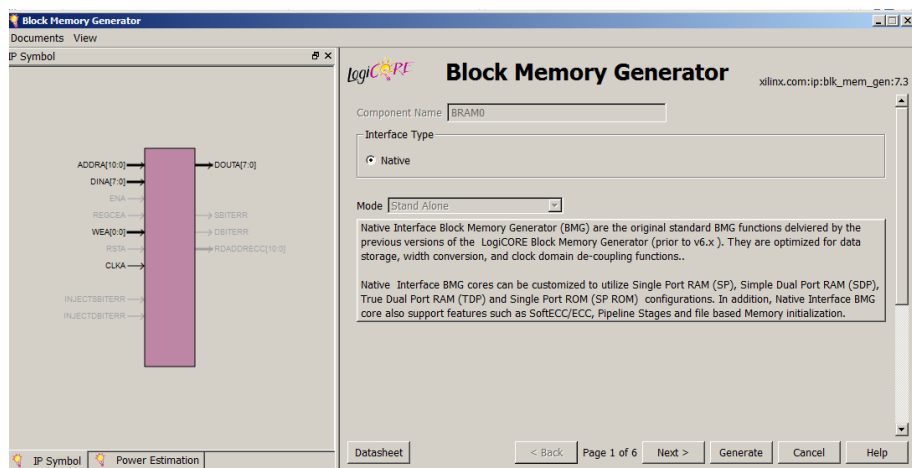


Figure 6.8: BRAM settings

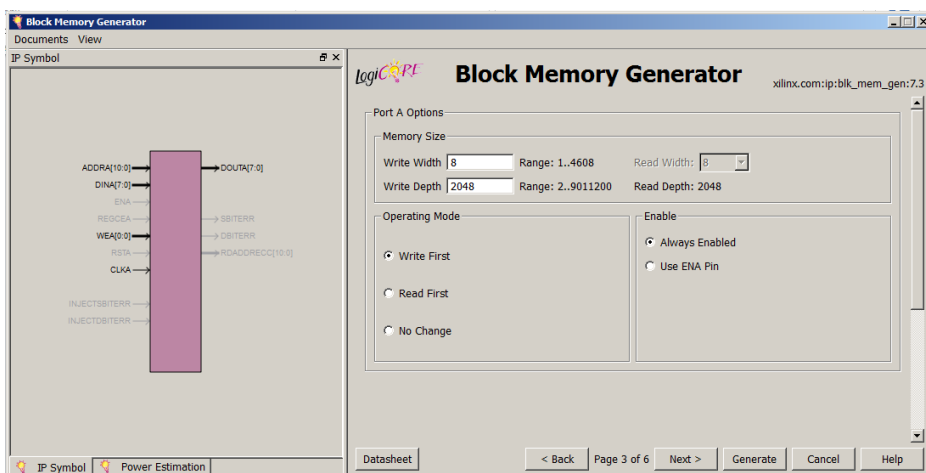


Figure 6.9: 8x2048 size BRAM

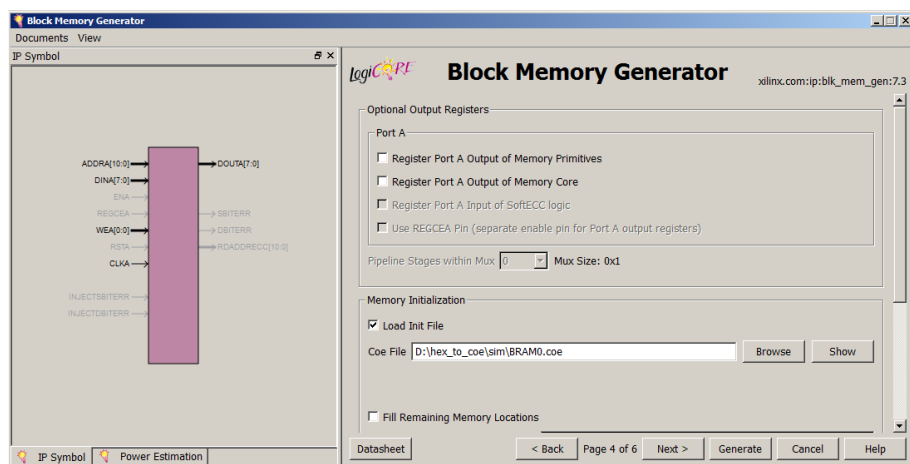


Figure 6.10: BRAM file path

Press generate button to make BRAM.

For clock settings we use DCM IP core. Set Input clock frequency 50 MHz and set output clock frequency. The settings should be as shown in the figure 6.11

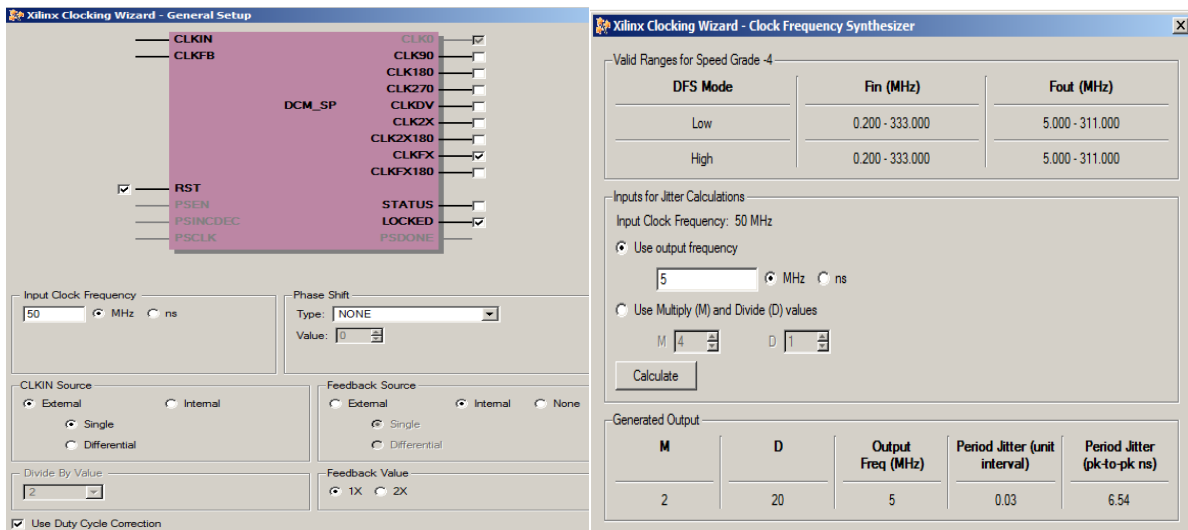


Figure 6.11: DCM clock frequency settings

6.2.1 UART pulse counter

In order to show proper working of M0 core, we need to make a file of 'uart_pulsecounter.v' file. As it is already discussed that M0 core sends out data on UART. We can observe that data by attaching serial port with our board and using any standard serial port GUI we can observe data transmitted by M0. Here we are using LEDs for same purpose. In 'uart_pulsecounter.v' file we are counting 210 pulse transitions from 1 to 0. After that we are toggling a LED D13 on demo board. UCF for same function is as shown in the figure 6.12

```

9  NET "XTAL1" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
10 NET "XTAL1" PERIOD = 20.0ns HIGH 40%;
11
12
13
14
15
16
17
18 # ==== Discrete LEDs (LED) ====
19 # These are shared connections with the FX2 connector
20 NET "P1<5>" LOC = "R14" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
21 NET "uart2_txd_inv" LOC = "C3" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
22 NET "uart2_txen" LOC = "E6" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
23 NET "test_logic" LOC = "D6" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
24 NET "uart_slow_pulse" LOC = "D13" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
25 #NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
26 #NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
27 #NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
28
29
30 # ==== Slide Switches (SW) ====
31 NET "NRST" LOC = "L13" | IOSTANDARD = LVTTTL | PULLUP ;
32 #NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTTL | PULLUP ;

```

Figure 6.12: UCF of M0 core

Top level M0 core implementation is shown in figure 5.13

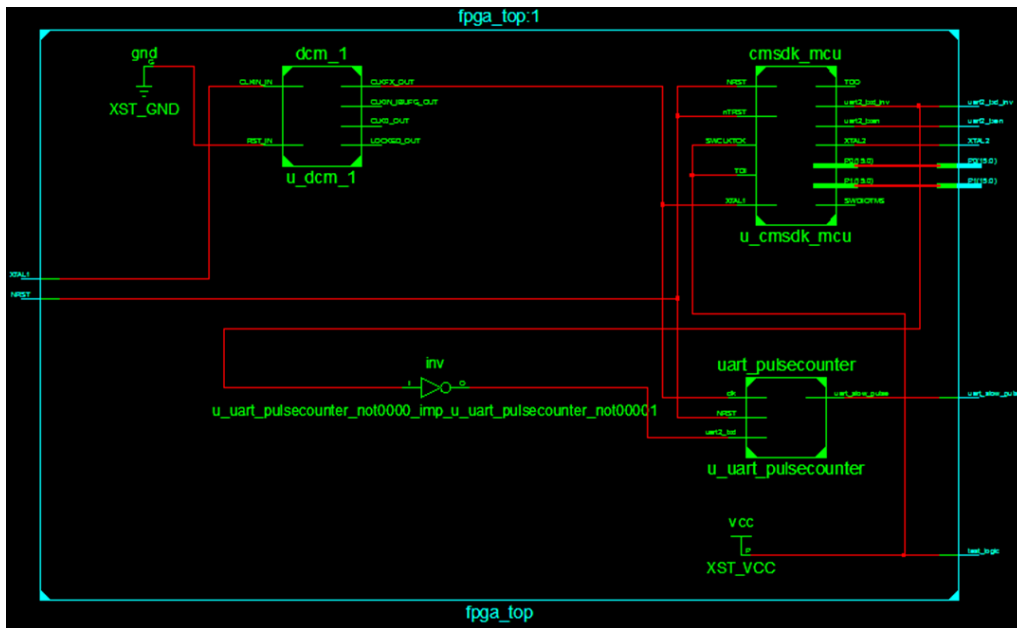


Figure 6.13: top level M0 core implementation

After code compilation and generation of file, we can download same file in FPGA and results as shown in figure 6.14 at HyperTerminal.

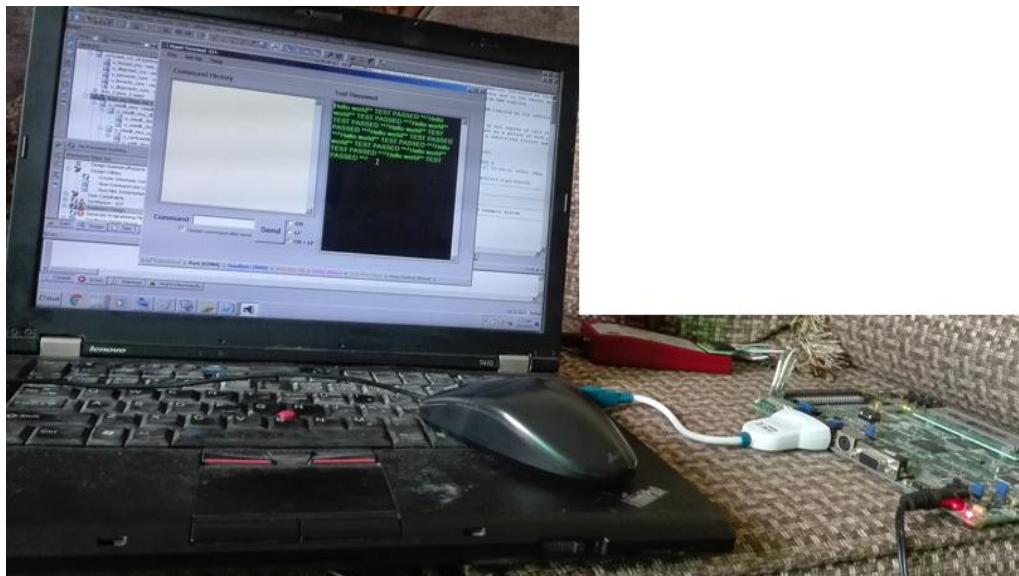


Figure 6.14: M0 core implementation on FPGA and UART functionality

Chapter 8: CONCLUSION & FUTURE WORKS

8.1 Conclusion

The achievement and conclusion of this work is that cortex Mo customized soft core is implemented in a smart FPGA which is upward compatible with upper cores M1, M2 ...,also which can be used as customized smart ASIC (Application Specific Integrated Circuit) for various applications like coprocessor in WSN ,crypto, parallel processing and in communication .

ARM Cortex Mo soft-core is designed, synthesized, and coded with code memory of FPGA using Advance Microcontroller Bus Architecture (AMBA Lite) and APB UART is functional additionally. In Mo soft core UART (FPGA) is working via which serial communication is possible. Error free serial data communication via customized soft core in FPGA is achieved.

Both hardware and software implementation in work and practical demonstration giving real successful implementation of customized Mo core which can be further configured according to requirement and application.

8.2 Future Work

In future works, the implemented cortex Mo core will be used for

- Encryption,
- Wireless sensor node and
- Security purpose.

For that purpose, in order to increase the processors capacity, other peripherals can be connected to the AMBA bus.

ZYNQ Linux operating system can be run over this processor that will make possible, a small footprint design, to get Linux implementation in.

References

- [1] ARM Ltd, "ARM DDI 0419C ARMv6-M Architecture Reference Manual", September 2010.
- [2] ARM Ltd, "ARM IHI 0033A AMBA 3 AHB-Lite Protocol V.1 Specification", June 2006.
- [3] Calix A Recado , Sankaran Rajesh "On the feasibility of an embedded machine learning processor for intrusion detection," IEEE international conference on Big Data, 2016.
- [4] Maas Martin, Love Eric, Stefanov Emil, Tiwari Mohit, Song Dawn "PHANTOM: Practical Oblivious Computation in a Secure Processor", university of California Berkeley.
- [5] Wang yi, Shilong Lu "Design and implementation of a SOC based security coprocessor and program protection mechanism for WSN".
- [6] ARM Ltd, "AT510-DC-80001-r0p0-00-rel0 ARM Cortex M0 Design Start Release Note" August 2010.
- [7] ARM Ltd, "ARM DDI 0432C Cortex M0 Revision r0p0 Technical Reference Manual", November 2009.
- [8] ARM Ltd, "ARM DUI 0497A Cortex M0 Devices Generic User Guide", October 2009.
- [9] Xilinx, "DS312 Spartan-3E FPGA Family: Datasheet", August 2009.
- [10] Digilent, "Digilent Spartan 3E Starter Kit Reference Manual", June 2008.
- [11] Costan victor , Lebedev Iliia, and Davedas Srinivas , *MIT CSAIL* "Sanctum: Minimal Hardware Extension for Strong Software Isolation".
- [12] "Private Core," <http://www.privatecore.com/>.
- [2] J. Agat, "Transforming out Timing Leaks," in POPL, 2000.
- [13] A. Askarov, D. Zhang, and A. C. Myers, "Predictive black-box mitigation of timing channels," in CCS, 2010.
- [14] A. Aviram, S. Hu, B. Ford, and R. Gummadi, "Determinating Timing Channels in Compute Clouds," in CCSW, 2010.

- [15] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avizienis, J. Wawrzynek, and K. Asanovic, "Chisel: Constructing Hardware in a Scala Embedded Language," in DAC, 2012.
- [16] K.-M. Chung, Z. Liu, and R. Pass, "Statistically-secure oram with $\tilde{O}(\log^2 n)$ overhead," <http://arxiv.org/abs/1307.3699>, 2013.
- [17] B. Coppens, I. Verbauwhede, K. D. Bosschere, and B. D. Sutter, "Practical Mitigations for Timing-Based-Channel Attacks on Modern x86 Processors," in SP, 2009.
- [18] J. Devietti, B. Lucia, L. Ceze, and M. Oskin, "DMP: Deterministic Shared Memory Multiprocessing," in ASPLOS, 2009.
- [19] C. W. Fletcher, M. v. Dijk, and S. Devadas, "A Secure Processor Architecture for Encrypted Computation on Untrusted Programs," in STC, 2012.
- [20] C. Gentry, K. Goldman, S. Halevi, C. Jufta, M. Raykova, and D. Wichs, "Optimizing oram and using it efficiently for secure computation," in PETS, 2013.
- [21]. 1. Daemen and V. Rijmen, "AES proposal: Rijndael" (1999).
- [22] [http://www.chipcon.com/files/CC2431 Brochure.pdf](http://www.chipcon.com/files/CC2431%20Brochure.pdf), 2008-03-20.
- [23]. [http://www.jennic.com/files/support files/JN-DSJN5121-1 v8.pdf](http://www.jennic.com/files/support%20files/JN-DSJN5121-1%20v8.pdf), 2008-03-20
- [24]. R.L. Rivest, "The RC5 encryption algorithm", in: Workshop on Fast Software Encryption (1995) pp. 86-96.
- [25]. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. "SPINS: Security Protocols for Sensor Networks". In Mobile Computing and Networking 2001, Rome, Italy, 2001.
- [26]. Karlof, N. Sastry, and D. Wagner. "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks". In ACM Conference on Embedded Networked Sensor Systems (SenSys '04), November 2004.

- [27]. Xiaoguang Niu, Yanmin Zhu, Li Cui, Lionel M. Ni, "FKM: A Fingerprint-based Key Management Protocol for SoC-based Sensor Networks", in Wireless Communications and Networking Conference, 2009. WCNC 2009. IEEE, 2009
- [28]. X. Huang, Z. Zhao , L Cui. "EasiSOC: Towards Cheaper and Smaller". Mobile Ad-hoc and Sensor Networks 2005(MSN'05). New York: Springer-Verlag, 2005. 229-238.
- [29]. Shilong Lu, Xi Huang, Li Cui, Ze Zhao, Dong Li. "Design and Implementation of an ASIC-based Sensor Device for WSN Applications". IEEE Transactions on Consumer Electronics, 55(4): 1959-1967, November 2009.
- [30]. Yi Wang, Shilong Lu, and Li Cui. "A Low Power Low BER Digital Base-band Design for WSN SoC". Acta Electronica Sinica, 38(2A): 123-127, Feb. 2010
- [31] O. Goldreich, "Towards a Theory of Software Protection and Simulation by Oblivious RAMs," in STOC, 1987.
- rr[32] O. Goldreich and R. Ostrovsky, "Software Protection and Simulation on Oblivious RAMs," J. ACM, 1996.
- [33] M. T. Goodrich and M. Mitzenmacher, "Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation," in ICALP, 2011.
- [34] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia, "Privacy-preserving Group Data Access via Stateless Oblivious RAM Simulation," in SODA, 2012.
- [35] Y. Gu, Y. Fu, A. Prakash, Z. Lin, and H. Yin, "OS-Sommelier: Memory-only Operating System Fingerprinting in the Cloud," in SoCC, 2012.
- [36] A. Haeberlen, B. C. Pierce, and A. Narayan, "Differential Privacy Under Fire," in USENIX Security, 2011.
- [37] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest We Remember: Cold-boot Attacks on Encryption Keys," Commun. ACM, vol. 52, no. 5, 2009.